# Head First EJB™

## (中文版)

### 助你通过Sun认证业务组件开发人员考试

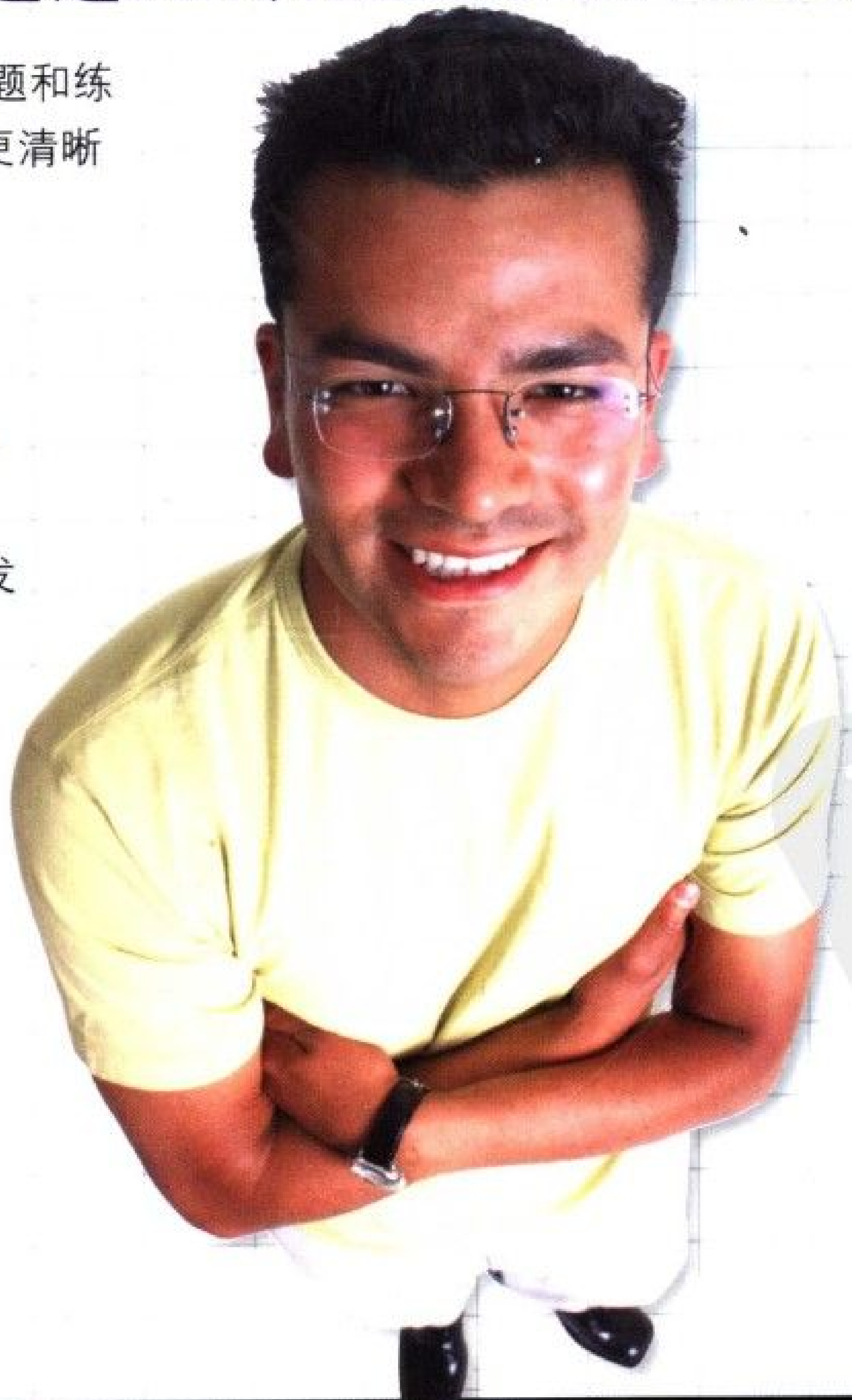


了解Lucy为什么喜欢离我们很遥远？




提供200余个实际的模拟测验题让你自测

用42个EJB问题和练习让你思路更清晰



由实际的 SCBCD 考试开发人员编写




偷听bean之间的对话，发现它们隐藏的秘密




提醒你的大脑，当心考试中的陷阱和 EJB “圈套（pitfall）”



利用旁门左道发现实战提示



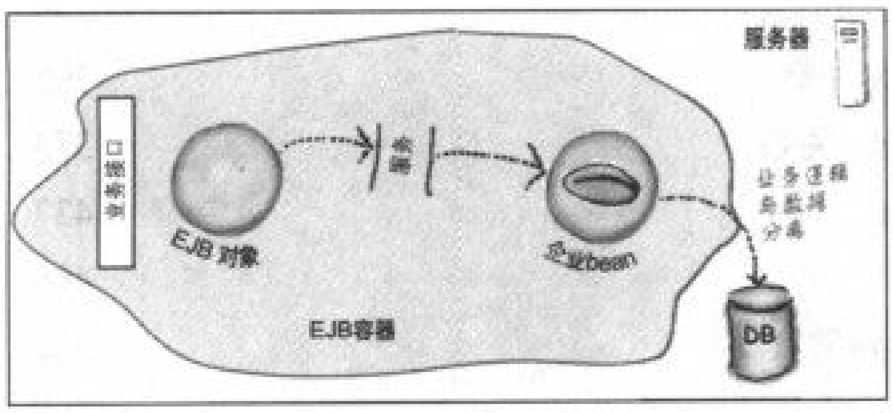
发现成功通过考试的人对EJB都了解些什么

Kathy Sierra & Bert Bates 著
陈学峰 等译

O'REILLY® 中国电力出版社

# 1 欢迎来到EJB世界

**Enterprise JavaBeans（EJB）并不难。** 不错，想想看，如果让你自己动手写一个可扩展性、事务性、安全性和并发性兼备的企业服务器，该是多么困难，所以至少从这一点来看，相比之下还是使用EJB简单得多。这一章我们将开发、部署并运行一个EJB应用，然后再深入讨论它的细节。在这一章的学习过程中，我们将分析怎样使用EJB，EJB有什么好处，它有哪些特点，另外，我们还会（简要地）谈到EJB容器如何工作。

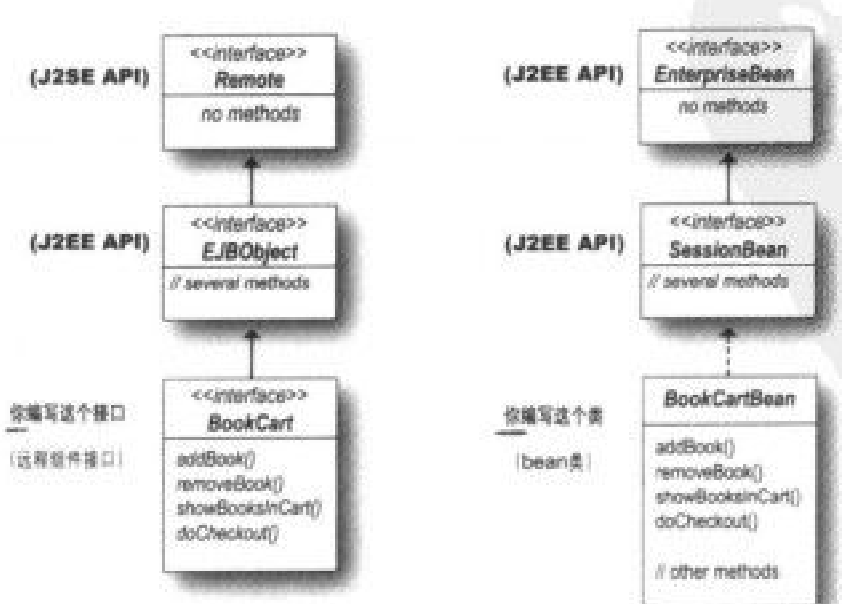




| | |
|---|---|
| 考试大纲 | 2 |
| EJB到底是什么？ | 3 |
| 不再为开发商所困！ | 5 |
| 它是怎么工作的？ | 7 |
| 幕后的动作…… | 8 |
| bean的三种类型 | 11 |
| 顾问bean | 15 |
| 构建bean要做的5件事 | 16 |
| EJB角色和职责 | 26 |
| 教程 | 28 |
| 模拟测验 | 59 |


# 2 EJB体系结构

**EJB是基础架构。** 组件是一些构建模块。利用EJB，除了简单的小应用外，还可以构建非常大的应用。这包括各种规模的应用，从复杂的Victoria's Secret后端应用到简单的CERN文档处理系统，都可以利用EJB来构建。但是，作为一个体系结构，要有这样高度的灵活性和可扩展性以及这么强大的功能，它肯定不是泛泛之辈。这种体系结构的基础往往是一个分布式编程模型……

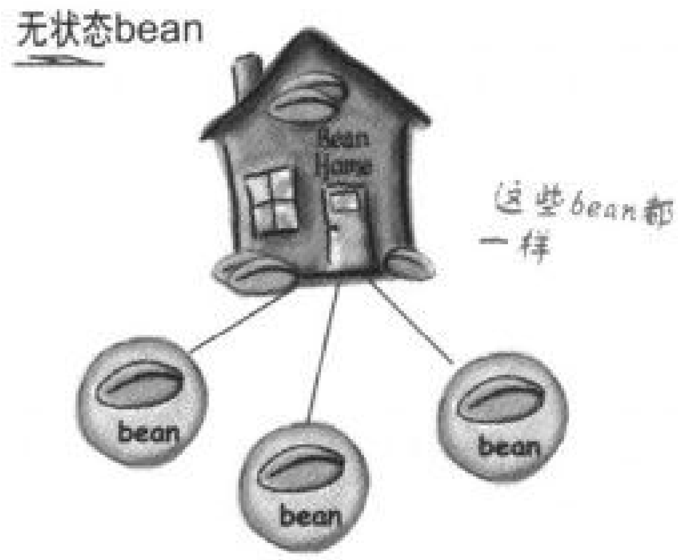




| | |
|---|---|
| 考试大纲 | 62 |
| 建立一个远程方法调用 | 64 |
| 关于参数和返回值 | 67 |
| 客户通过远程接口调用业务方法 | 79 |
| EJB如何使用RMI | 81 |
| 远程对象不是bean，它是bean的保镖 | 82 |
| 体系结构概述：会话bean | 98 |
| 体系结构概述：实体bean | 99 |
| 体系结构概述： 创建有状态会话bean | 100 |
| 体系结构概述： 创建无状态会话bean | 101 |
| 体系结构概述：消息驱动bean | 106 |
| bean的组织 | 108 |

# 3 表现你自己

**不能让bean太隐密了。** 客户需要看到你有什么（消息驱动bean除外，它没有客户视图）。Advice Bean在它的组件接口里提供了getAdvice()方法，组件接口就是声明业务方法的地方。不过，这并不是客户看到的全部。要记住，Advice接口扩展了EJBObject，这个EJBObject接口又有它自己的方法。这些方法客户也能看到，而且能调用。Home接口也一样。




对于来自同一个home的无状态会话bean，isIdentical()总是返回*true*，甚至是两个不同的bean。


| | |
|---|---|
| 考试大纲 | 112 |
| 客户到底想要什么？ | 113 |
| 什么是JNDI？ | 116 |
| PortableRemoteObject.narrow()（奇特的类型强制转换） | 121 |
| 为会话bean编写远程home接口 | 125 |
| 好在我们拿到了句柄（在线购物是要花时间的） | 139 |
| 对于本地客户接口，哪些方法有意义？ | 148 |
| 为什么这么多删除方法？ | 151 |
| 远程接口和本地接口的比较 | 154 |
| 远程和本地方法的参数 | 163 |
| 模拟测验 | 168 |


# 4 当个会话bean

**会话bean要创建和删除。** 如果你是一个无状态bean，那你很幸运。因为有状态bean的一生都操纵在一个无情的客户手上。有状态bean是在一个客户的坚持下创建的，它的存在只是要为这个客户服务。不过，啊哈，相比之下，无状态bean的生活真是帅呆了！泳池、阳伞、饮料，一切都这么美好，这里没有乏味、没有单调，因为你要面对那么多形形色色的客户。





| | |
|---|---|
| 考试大纲 | 174 |
| 容器回调，bean一生中的特殊时刻 | 181 |
| bean创建 | 188 |
| 业务方法中可能做的bean工作 | 199 |
| 钝化：有状态bean实现可扩展性的机会 | 200 |
| bean删除 | 208 |
| 编写一个会话bean：这是bean提供者的任务 | 230 |
| SessionContext：你需要它，超过了它需要你 | 240 |
| 模拟测验 | 244 |

# 5 实体都是持久的

**实体bean会一直保存。** 实体bean一直存在。它们是底层持久存储库中某个东西的对象表示（可以把底层持久存储库想成是数据库，因为大多数实体bean表示的就是一个关系数据库中的某个东西）。如果有一个Customer实体bean，那么可能有一个bean表示实体Tyler Durden，ID为#343，另一个bean表示实体Donny Darko，ID是#42。这两个bean分别表示两个实际的实体。实体bean只是持久存储库中已经存在的某个东西的一种具体实现。








| | |
|---|---|
| 考试大纲 | 260 |
| 什么是实体bean? | 261 |
| 客户视角看到的实体bean | 265 |
| 一个非常简单的Customer实体bean | 268 |
| 实体bean远程组件接口 | 270 |
| 实体bean远程home接口 | 273 |
| 客户到底想从一个实体bean home得到什么? | 274 |
| 解决之道：home业务方法 | 278 |
| 会话bean create()与实体bean create() | 281 |
| 会话bean remove()与实体bean remove() | 282 |
| 实体/bean/实例的死亡 | 285 |
| 模拟测验 | 288 |


# 6 当个实体bean

**实体bean就是演员。** 只要还活着，实体bean就要么待在池子里，要么扮演着某个人，这是底层持久存储库里的一个人。换句话说，也就是数据库里的一个实体。当bean扮演某个实体时，这个bean必须与底层实体保持同步。如果bean假装是某个人，比方说是Audrey Leone，要是有人在数据库里降低了Audrey的信贷限额，但是忘了告诉bean……

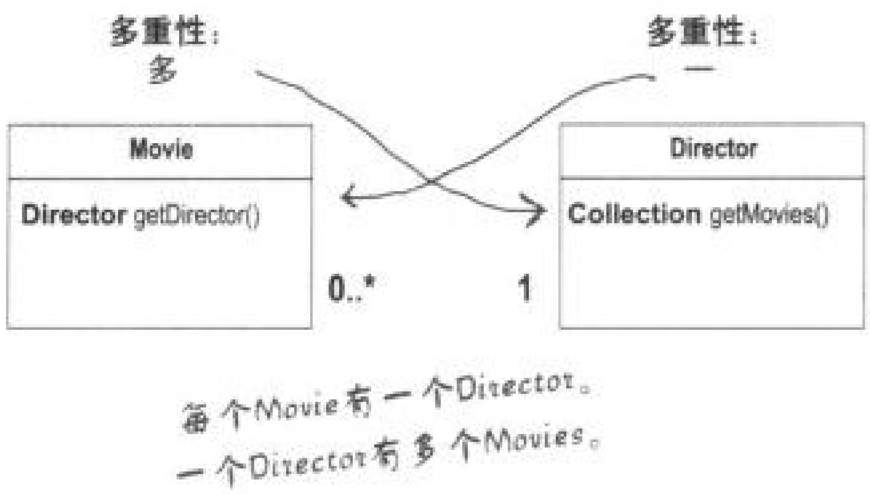




| | |
|---|---|
| 考试大纲 | 296 |
| 实体bean的真正强大之处在于同步 | 298 |
| 容器托管持久存储与bean托管持久存储 | 303 |
| EntityBean接口增加了新的容器回调 | 310 |
| 编写CMP实体bean | 313 |
| 对象身份：主键 | 332 |
| 查找方法 | 339 |
| Home业务方法 | 345 |
| 模拟测验 | 362 |

# 7 bean之间的关系

**实体bean需要关系。**一个订单需要一位顾客。一件商品需要一个订单。一个订单需要多件商品。实体bean可以有容器托管关系（container-managed relationships，CMR），由容器负责几乎所有一切。生产了一个新商品吗？如果这个商品与一个订单关联，你要求顾客给你出示他的订单时，他的订单就会显示出这个新商品。还有最好的一点，这就是可以使用EJB-QL来编写可移植查询。

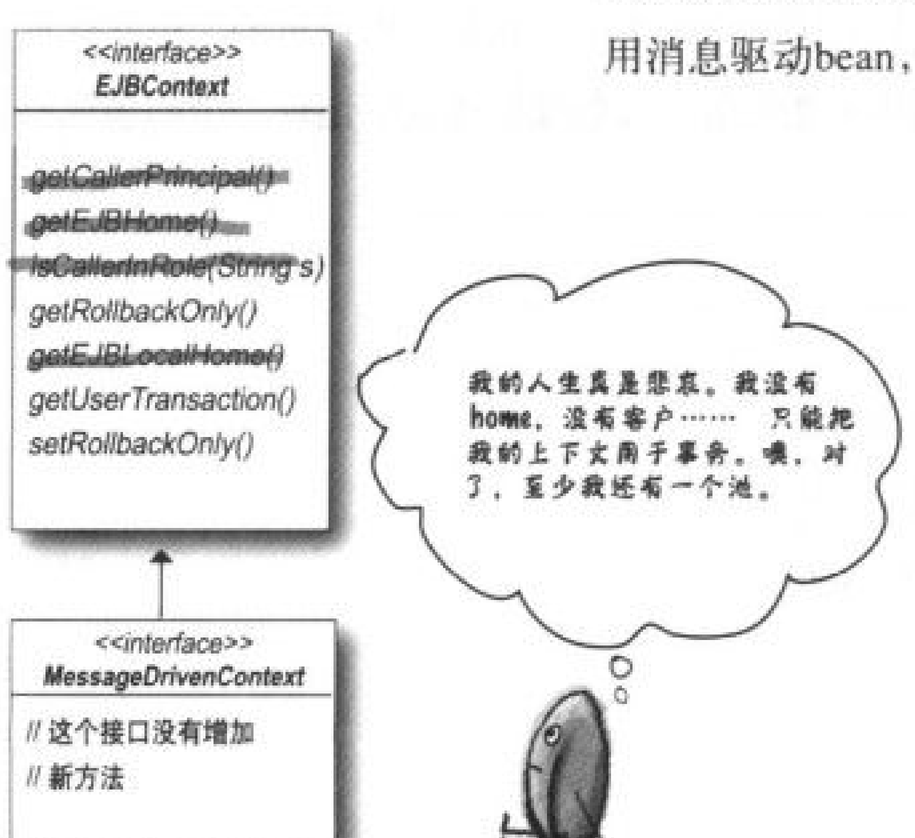




| | |
|---|---|
| 考试大纲 | 374 |
| 关系 | 378 |
| 多重性 | 380 |
| CMP和CMR字段 | 383 |
| 级联删除可以传播 | 393 |
| MovieBean的EJB-QL | 402 |
| SELECT和FROM是必不可少的! | 409 |
| WHERE子句 | 411 |
| Collection不能调用bark()! | 414 |
| BETWEEN、IN、IS EMPTY和LIKE表达式 | 416 |
| 关系指派 | 421 |
| 模拟测验 | 425 |


# 8 得到消息

**接收消息很有意思。**虽然不是特别的好玩，比如说，比不上收到eBay给你的一个真正的Smurf车灯包裹，但确实很有意思，也很高效。假设你向eBay提交了一个订单，在包裹递送到之前你就不能离开家（否则就会错过）。会话bean和实体bean就是这样，但是利用消息驱动bean，客户发送完消息后就可以走开。





| | |
|---|---|
| 考试大纲 | 438 |
| 编写消息驱动bean: bean提供者的任务 | 447 |
| 消息驱动bean的完整部署描述文件 | 449 |
| 主题和队列 | 450 |
| MessageDrivenContext | 455 |
| 消息确认 | 458 |
| 模拟测验 | 463 |

# 9 原子时代

**事务可以保护你。**利用事务，可以冒点险。你可以尝试做些事情，因为你知道即使过程中出了意外，也可以假装什么事情都没有发生。一切又能恢复到从前。EJB中的事务是一大妙笔，你可以部署一个有定制事务行为的bean，而不用去动这个bean的源代码！但是如果需要，也确实可以编写事务代码。





| | |
|---|---|
| 考试大纲 | 470 |
| ACID测试 | 472 |
| EJB中是如何做到的 | 474 |
| 事务传播 | 475 |
| 怎么建立（或得到）一个事务？ | 476 |
| 两个接口里都有setRollbackOnly() | 487 |
| BMT可能是一个很糟糕的想法。BMT会破坏bean重用 | 490 |
| 容器托管事务 | 491 |
| 属性的作用 | 492 |
| 必须标记属性的方法（CMT bean） | 498 |
| “未指定的事务上下文” | 499 |
| CMT的部署描述文件示例 | 503 |
| SessionSynchronization“特殊时刻” | 512 |
| 模拟测验 | 516 |


# 10 bean要是坏了

**要做最坏的打算。**尽管你竭尽全力，但事情还是有可能出错。也许很可怕，很悲惨，错得很厉害。你要保护自己，而且可能别人还依赖着你。不能只是因为一个bean抛出一个异常就使得整个程序都陷于崩溃。应用必须继续下去。悲剧是不能避免的，但是你可以做好准备。你要知道哪些可以恢复，哪些不能恢复，另外要知道如果出现一个问题该归咎于谁。

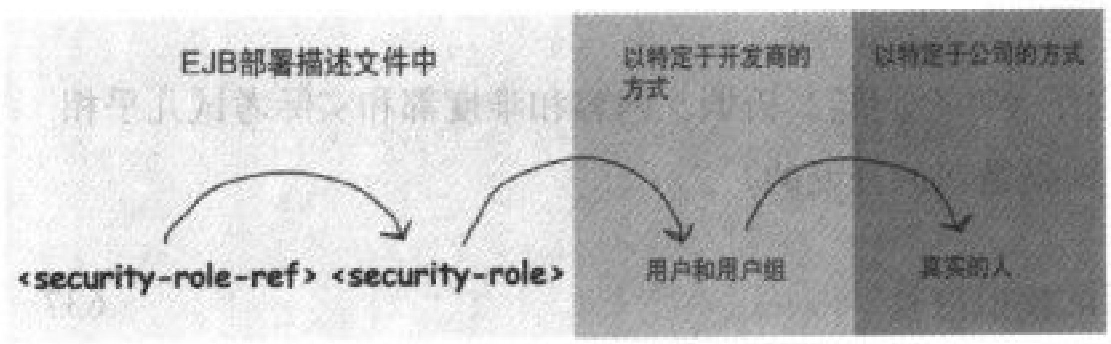




| | |
|---|---|
| 考试大纲 | 526 |
| EJB中，异常有两类 | 532 |
| 对于一个应用异常，容器会…… | 533 |
| 对于一个系统异常，容器会…… | 534 |
| RemoteException与EJBException | 539 |
| bean提供者的职责 | 541 |
| 5个标准EJB应用异常 | 548 |
| 常用的系统异常 | 551 |
| 模拟测验 | 563 |

# 11 保护你的秘密

**守住你的秘密。**安全所涉及的就是认证和授权。首先，你必须证明你的身份，然后我们会告诉你可以做哪些事情。在EJB中，安全很简单，因为你只需要处理授权，确定谁能调用bean的哪些方法。不过有一个问题要注意……如果你是一个bean提供者或者应用组装人员，你可能不知道最终的用户到底是什么人！





考试大纲 570

EJB中如何做到安全 573

应用组装人员的任务：访问控制 574

定义方法许可 578

部署人员的任务：把实际的人映射到抽象的角色 583

类级安全与实例级安全 586

使用程序方式实现安全来定制方法 587

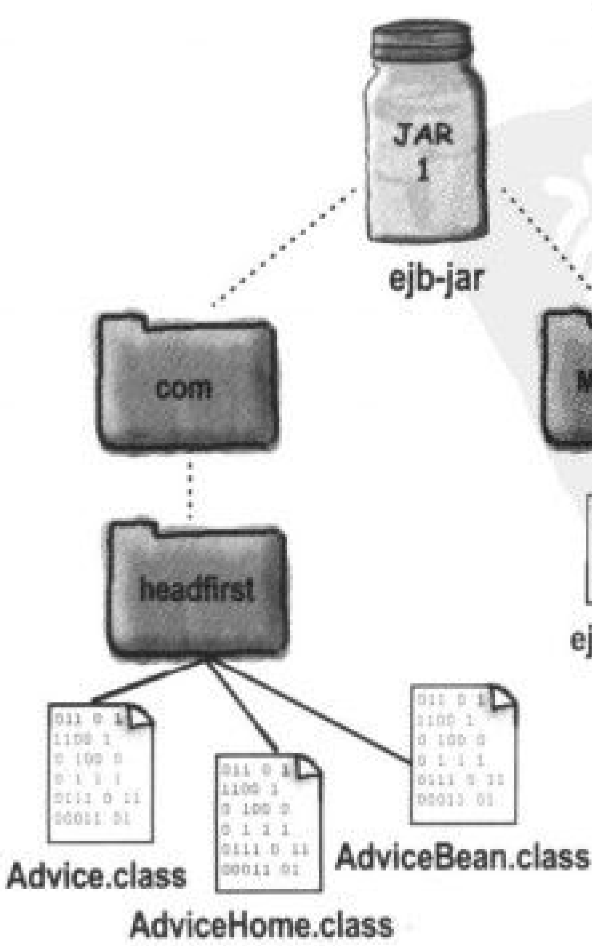
使用 <run-as>安全身份假装成别人在调用…… 591

采用 <run-as>的安全上下文传播 592

模拟测验 593


# 12 部署的快乐

**bean简直让你费尽力气。**你要编写代码、完成编译，进行测试。这个过程反反复复地做了无数次。你最不希望的就是，仅仅因为部署配置中的一个小小的改变，就要对已经测试过的源代码再做改动。也许是数据库的名字变了，另外，如果你根本拿不到源代码怎么办？EJB通过可定制的部署描述文件和bean的一个特殊环境可以支持bean重用。





考试大纲 600

bean的特殊位置java:comp/env 602

创建和使用子上下文 609

bean提供者和应用组装人员的职责 617

部署人员的职责 618

记住谁做什么 619

哪些API是EJB 2.0 保证支持的？ 621

ejb-jar中必须有什么？ 624

编程限制 625

模拟测验 627

# A 附录

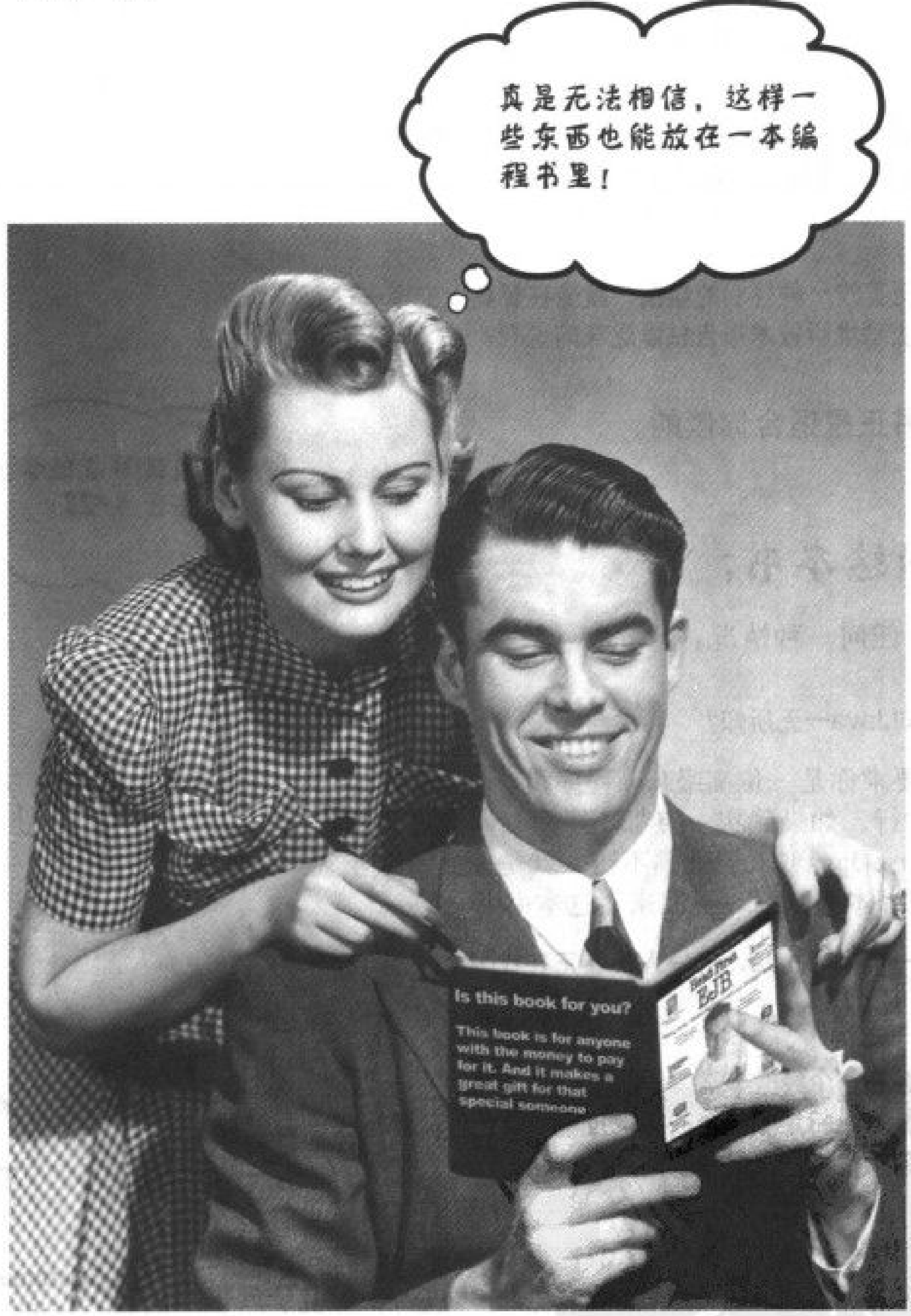



**最后的模拟测验。**终于到它了。70个问题。语调、内容和难度都和实际考试几乎相同。这一点我们很清楚（因为考试就是我们开发的）。

最后的模拟测验 637

# i 索引 685

# 引子




有一个问题真是听得把我们的耳朵都磨出茧了，这就是："你们为什么要把这样一些东西放在一本编程书里呢？"这一节正是要回答这个问题。

# 谁适合读这本书？

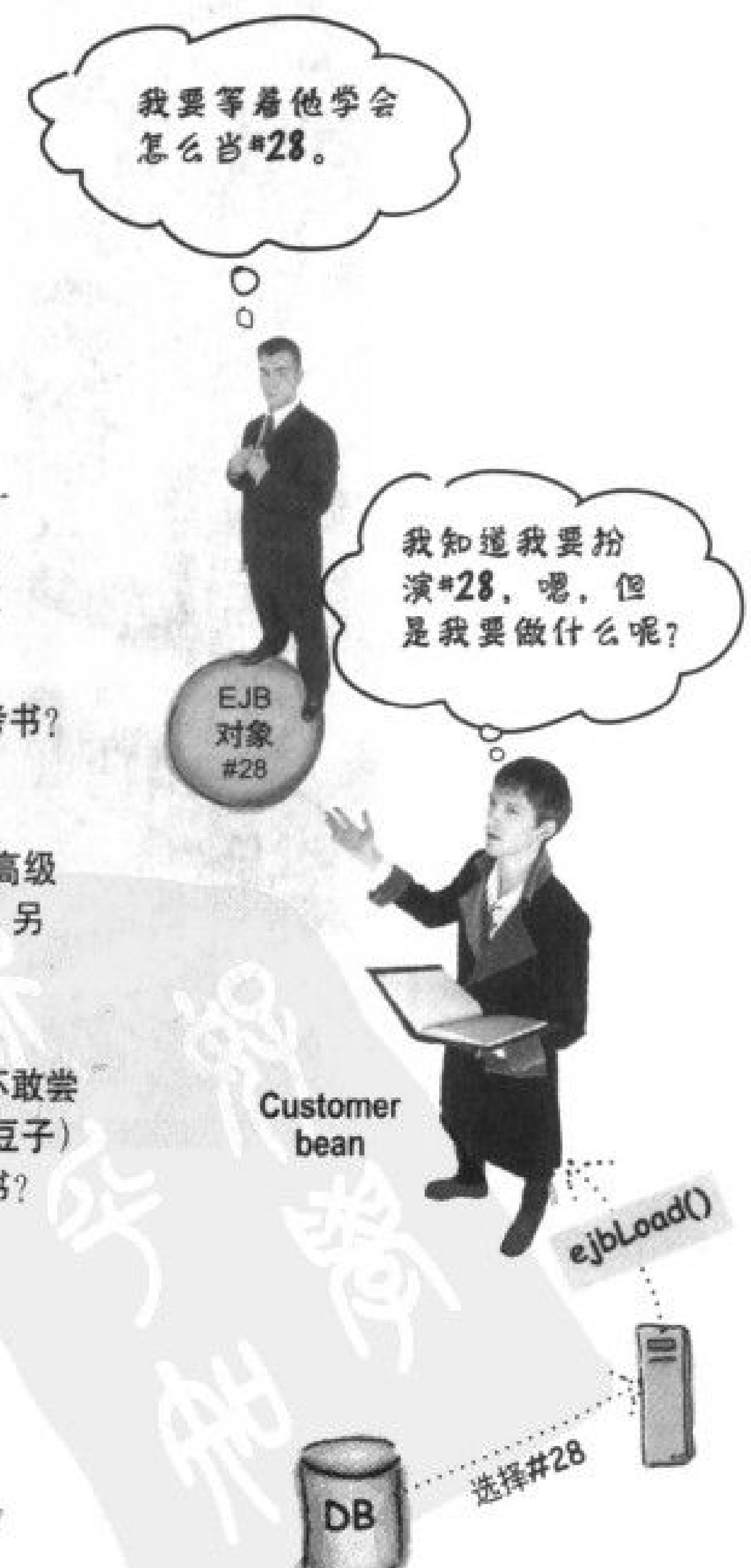
如果对下面的所有问题你都能肯定地回答“是”：

1. 你懂Java吗？（不过不要求精通。）

2. 你是不是想学习并通晓EJB，把它牢牢记住，还想顺利通过SCBCD考试，而且能够真正开发业务组件？

3. 你是不是更喜欢一种轻松的氛围，就像在餐桌上交谈一样，而不愿意被动地听技术报告枯燥乏味的说教？

**那么，这本书正是适合你读的。**




# 谁暂时还不适合读这本书？

如果满足下面任何一种情况：

1. 你是不是对Java一无所知？

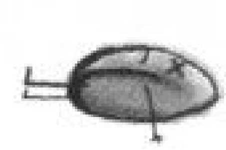
   （虽然不要求你是一位高级Java程序员，但起码要有一些经验才行。如果确实一点都不了解Java，先买一本《Head First Java》看看吧，不错，就是今天，事不宜迟，看完那本书以后，再回来看这本书吧。）

2. 你是不是一个很棒的Java开发人员，正在找一本EJB的参考书？

3. 你本身已经是一个经验丰富的J2EE专家，需要了解一些超高级的服务器技术、特定服务器的有关技术以及企业设计模式，另外希望得到大量复杂、健壮的真实代码。

4. 你是不是对新鲜事物都畏手畏脚？只喜欢简单的道道，而不敢尝试把条纹和格子混在一起？你是不是觉得，如果把bean（豆子）都拟人化了，这样的一本书肯定不是一本正儿八经的技术书？

**那么，太遗憾了，这本书不适合你。**



[来自市场的声音：不是说只要买得起，就都能看这本书吗？是谁把这种说法去掉了？干嘛这样限制我们的读者群？]

## 我们知道你在想什么。

"这算一本正儿八经的编程书吗？"

"这些图是干什么的？"

"我真地能这样学吗？"

"这简直就是个比萨嘛！"



## 我们也知道你的大脑在想什么。




你的大脑总是渴求一些新奇的东西。它一直在搜寻、审视、期待着不寻常的事情发生。大脑的构造就是如此，正是这一点才让我们不至于固步自封，能跟着时代前进。

如今，一般是不太可能被老虎吃掉的。然而，你的大脑还是一直在注意着周围是否有潜伏的老虎。只不过你自己没有意识到而已。

但是我们每天都会遇到许多按部就班的事情，这些事情很普通，对于这样一些例行的事情或者平常的东西，你的大脑又是怎么处理的呢？它的做法很简单，就是不让这些平常的东西妨碍大脑真正的工作，那么什么是大脑真正的工作呢？这就是记住那些确实重要的事情。它不会费心地去记乏味的东西；就好像大脑里有一个筛子，这个筛子会筛掉"显然不重要"的东西，如果遇到的事情枯燥乏味，这些东西就无法通过这个筛子。

那么你的大脑怎么知道到底哪些东西重要呢？打个比方，假如你某一天外出旅行，突然一只大老虎跳到你面前，此时此刻，你的大脑里会发生什么呢？

看到这只大老虎，你的神经元会"开火"，情绪爆发，释放出一些化学物质。

好了，这样你的大脑就会知道……

**这肯定很重要！可不能忘记了！**




不过，假如你正待在家里，或者坐在图书馆里。这里很安全，很温暖，肯定没有老虎。你正在刻苦学习，准备应付考试。也可能想学一些比较难的技术，你的老板认为掌握这种技术需要一周时间，最多不超过十天。

这就存在一个问题。你的大脑很想给你帮忙。它会努力地把这些显然不太重要的内容赶走，保证这些东西不去侵占本不算充足的脑力资源。这些资源最好还是用来记住确实重要的事情。比如大老虎，再比如火灾险情。如果你曾经只是身着短衣裤被大雪围困，这件事肯定不会忘却，你的大脑会记住绝不要让这种情况再发生第二次。

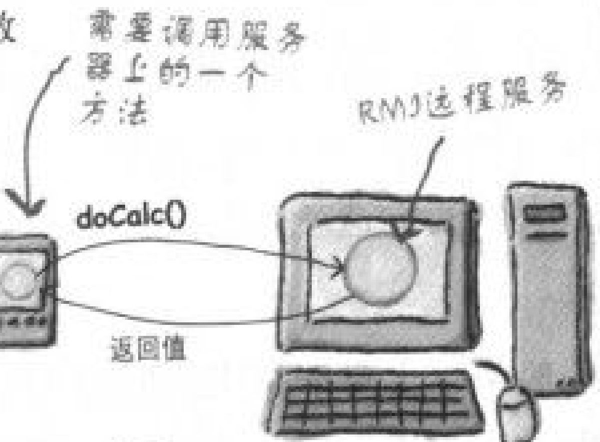
我们没有一种简单的办法来告诉大脑，"嘿，大脑，真是谢谢你了，不过不管这本书多没意思，也不管我对它是多么地无动于衷，但我确实希望你能帮助我把这些东西记下来。"

# 我们认为，“Head First”的读者就是要学习的人。

那么，怎么来学习呢？首先，必须了解，然后要保证自己确实不会忘记。这可不是填鸭式的硬塞。根据认知科学、神经生物学和教育心理学的最新研究，学习的途径相当丰富，绝非只是通过书本上的文字。我们很知道怎么让你的大脑兴奋起来。

**下面是一些Head First学习原则：**






**看得到。**与单纯的文字相比，图片更能让人记得住，通过图片，学习效率会更高（对于记忆和传递型的学习，甚至能有多达89%的效率提升）。而且图片更能让人看懂。以往总是把图片放在一页的最下面，甚至放在另外的一页上，与此不同，如果把文字放在与之相关的图片内部，或者在图片的周围写上相关文字，学习者的能力就能得到多至两倍的提高，从而能更好地解决有关的问题。

**采用一种针对个人的交谈式风格。**最新的研究表明，如果学习过程中采用一种第一人称的交谈方式直接向读者讲述有关内容，而不是用一种干巴巴的语调介绍，学生在学习之后的考试中成绩会提高40%。正确的做法是讲故事，而不是做报告。要用通俗的语言。另外不要太严肃。如果你面对着这样两个人，一个是你在餐会上结识的一位很有意思的朋友，另一个人学究气十足，喋喋不休地对你说教，在这两个人中，*你*会更注意哪一个呢？




**让学习的人想得更深。**换句话说，除非你很积极地让神经元活动起来，否则你的头脑里什么也不会发生。必须引起读者的好奇，促进、要求并鼓励读者去解决问题、得出结论、产生新的知识。为此，需要提出挑战，留下练习题和拓宽思路的问题，并要求读者完成一些实践活动，让左右脑都开动起来，而且要利用到多种思维。




```
abstract void roam();
```

没有方法体！
用一个分号结束。

**引起读者的注意，而且要让他一直保持注意。**我们可能都有过这样的体验，“我真的想把这个学会，不过看过一页后实在是让我昏昏欲睡”。你的大脑注意的是那些不一般、有意思、有些奇怪、抢眼的、意料之外的东西。学习一项有难度的新技术并不一定枯燥。如果学习过程不乏味，你的大脑很快就能学会。

**影响读者的情绪。**现在我们知道了，记忆能力很大程度上取决于所记的内容对我们的情绪有怎样的影响。如果是你关心的东西，就肯定记得住。如果让你感受到了什么，这些东西就会留在你的脑海中。不过，我们所说的可不是什么关于男孩与狗的伤心故事。这里所说的情绪是惊讶、好奇、觉得有趣、想知道“什么……”，还有就是一种自豪感，如果你解决了一个难题，学会了所有人都觉得很难的东西，或者发现你了解的一些知识竟是那些自以为无所不能的傲慢家伙所不知道的，此时就会有一种自豪感油然而生。



# 元认知：有关思考的思考

如果你真地想学，而且想学得更快、更深，就应该注意你怎样才会专注起来。考虑自己是怎样思考的，并了解你的学习方法。

我们中间大多数人长这么大可能都没有上过有关元认知或学习理论的课程。我们想学习，但是很少有人教我们怎么来学习。

不过，这里可以做一个假设，如果你手上有这本书，你想学EJB，而且可能不想花太多时间。另外因为你要参加考试，所以需要记住你读到的所有内容。为此必须理解这些内容。要想最大程度地掌握这本书或其他任何一本书中介绍的知识，就要让你的大脑负起责来，要求它记住这些内容。




怎么做到呢？技巧就在于要让你的大脑认为你学习的新东西确实很重要，对你的生活有很大影响。就像老虎出现在面前一样。如若不然，你将陷入旷日持久的拉锯战中，虽然你很想记住所学的新内容，但是你的大脑却会竭尽全力地把它们拒之门外。

## 那么究竟怎样才能让你的大脑把EJB看作是一只饥饿的老虎呢？

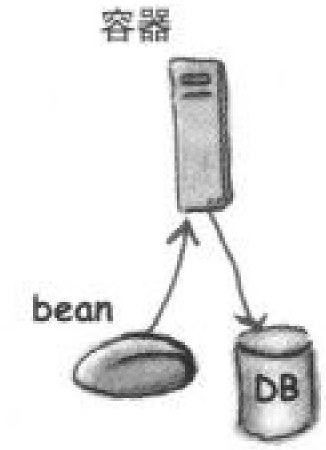
这有两条路，一条比较慢，很乏味。另一条路不仅更快，还更有效。慢方法就是大量地重复。你肯定知道，如果反反复复地看到同一个东西，即便再没有意思，你也能学会并记住。如果做了足够的重复，你的大脑就会说："尽管看上去这对他来说好像不重要，不过，既然他这样一而再、再而三地看同一个东西，所以我觉得这应该是很重要的。"

更快的方法是尽一切可能让大脑活动起来，特别是开动大脑来完成不同类型的活动。如何做到这一点呢？上一页列出的学习原则正是主要的一些可取做法，而且经证实，它们确实有助于让你的大脑全力以赴。例如，研究表明，把文字放在所描述图片的中间（而不是放在这一页的别处，比如作为标题，或者放在正文中），这样会让你的大脑更多地考虑这些文字与图片之间有什么关系，而这就会让更多的神经元开火。让更多的神经元开火 = 你的大脑更有可能认为这些内容值得注意，而且很可能需要记下来。

交谈式风格也很有帮助，当人们意识到自己在与"别人"交谈时，往往会更加关注，这是因为他们总想跟上谈话的思路，并能做出适当的发言。让人惊奇的是，大脑并不关心"交谈"的对方究竟是谁，即使你只是与一本书"交谈"，它也不会在乎！另一方面，如果写作风格很正式、干巴巴的，你的大脑就会觉得，这就像坐在一群人当中，被动地听人做报告一样，很没意思，所以不必在意对方说的是什么，甚至可以打瞌睡。

不过，图片和交谈风格还只是开始而已，能够做的还有很多。

# 我们是这么做的：




我们用了很多**图**，因为你的大脑更能接受看得见的东西，而不是纯文字。对你的大脑来说，一幅图顶得上1024个字。如果既有文字又有图片，我们会把文字放在图片当中，因为文字处在所描述的图片中间时，大脑的工作效率更高，倘若把这些描述文字作为标题，或者“湮没”在别处的大段文字中，那就达不到这种效果了。

我们采用了**重复手法**，会用不同方式，采用不同类型的媒体，运用多种思维手段来介绍同一个东西，目的是让有关内容更有可能储存在你的大脑中，而且能够在多个区中都有容身之地。




我们会用你**想不到**的方式运用概念和图片，因为你的大脑喜欢新鲜玩艺；在提供图和思想时，至少会含着一些情绪因素，因为如果能产生情绪反应，你的大脑就会投入更大的注意。而这会让你感觉到这些东西更有可能要被记住，其实这种感觉可能只是点**幽默**、让人**奇怪**或者比较感**兴趣**而已。

我们采用了一种针对个人的**交谈式风格**，因为当你的大脑认为你在参与一次交谈，而不是被动地听一场演示汇报时，它就会更加关注。即使你实际上在读一本书，也就是说在与书“交谈”，而不是真正与人交谈，但这对你的大脑来说并没有什么分别。




在这本书里，我们加入了40多个**实践活动**，因为与单纯的阅读相比，如果能实际做点什么，你的大脑会更乐于学习，更愿意去记。练习都是我们精心设计的，有一定的难度，但是确实能做出来，因为这是大多数人所希望的。




我们采用了**多种学习模式**，因为尽管你可能想循序渐进地学习，但是其他人可能希望先对整体有一个全面的认识，另外可能还有人只是想看一个代码例子。不过，不管你想怎么学，要是同样的内容能以多种方式来表述，这对每一个人都会有好处。




这里的内容不只是单单涉及左脑，也不只是让右脑有所动作，我们会让**你的左右脑**都开动起来，因为你的大脑参与得越多，你就越有可能学会并记住，而且能更长时间地保持注意力。如果只有一半大脑在工作，通常意味着另一半有机会休息，这样你就能更有效率地学习更长时间。

我们会讲故事，留练习，**从多种不同的角度**来看同一个问题，这是因为，如果要求大脑做一些评价和判断，它就能更深入地学习。

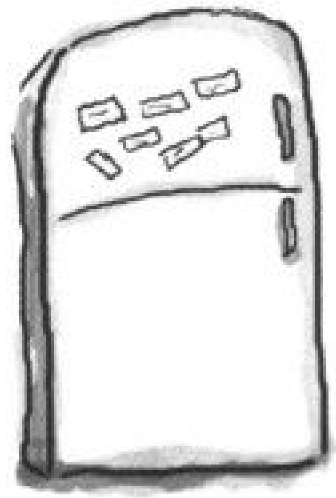



你会看到我们给出的一些**练习**，还要回答一些**问题**，这些问题往往不是直截了当就能做出回答的，通过征服这些挑战，你就能学得更好，因为让大脑真正做点什么的话，它就更能学会并记住。想想吧，如果只是在健身馆里看着别人流汗，这对于保持你自己的体形肯定不会有什么帮助，正所谓坐而观鱼，不如退而结网。不过另一方面，我们会竭尽所能不让你钻牛角尖，把劲用错了地方，而是能把功夫用在点子上。也就是说，你不会为搞定一个难懂的例子而耽搁，也不会花太多时间去弄明白一段艰涩难懂而且通篇行话的文字，我们的描述也不会太过简洁而让人无从下手。



我们用了**拟人手法**。在故事中，在例子中，还有在图中，你都会看到人的出现，这是因为你本身是一个人，不错，这就是原因。如果和人打交道，相对于东西而言，你的大脑会表示出更多的注意。

我们充分利用了80/20方法。我们认为，如果你真地要攻读EJB博士学位的话，这本书肯定不会是你惟一的EJB书。所以我们不打算方方面面都照顾到。这里只提供了你真正需要的东西。



# 可以用下面的方法让你的大脑就范

好了，我们该做的已经做了，剩下的就要看你自己的了。这些提示只是个开头：听一听你的大脑是怎么说的，弄清楚对你来说哪些做法可行，哪些做法不能奏效。

还可以做些新的尝试。

把这一页撕下来，贴到你的冰箱上。

---

1. **慢一点。你理解得越多，需要记得就越少。**
   不要光是看看就行了。停下来，好好想一想。书中提出问题的时候，你不要直接去翻答案。可以假想成真的有人在问你问题。你让大脑想得越深，就越有可能学会并记住。

2. **做练习，自己记笔记。**
   我们给你留了练习，但是如果这些练习的解答也由我们一手包办，那和有人替你参加考试有什么分别。不要只是坐在那里看着练习发呆。拿出笔来，写一写，画一画。大量研究都证实，学习过程中如果能实际动动手，这将改善你的学习。

3. **阅读“这里没有傻问题”部分。**
   顾名思义，这些问题可不是可有可无的旁注，它们绝对是核心内容的一部分！千万不要把它们跳过去不看。

4. **不要老待在一个地方看书。**
   起立，伸伸懒腰，四处走走，换把椅子，换个房间。这都能让你的大脑有点新的感觉，还能避免你的学习与一个特定的场所联系过紧。要记住，你肯定不会在自己的卧室里参加考试。

5. **上床睡觉之前不要再看别的书了，或者不要再看其他有难度的东西。**
   学习中有一部分是在你合上书之后完成的（特别是，要把学到的知识长久地记住，这往往无法在看书的过程中做到）。你的大脑也需要有自己的时间，这样才能再做一些处理。如果在这段处理时间内你又往大脑里灌输了新的知识，那么你刚才学的一些东西就会丢掉。

6. **要喝水，而且要多喝点水。**
   如果能提供充足的液体，你的大脑才能有最佳表现。如果缺水（可能在你感觉到口渴之前已经缺水了），学习能力就会下降。参加考试的时候，最好喝些啤酒，还可以来点更“有劲”的。

7. **大声地说出来。**
   说话可以刺激大脑的另一部分。如果你想看懂什么，或者想更牢地记住它，就要大声地说出来。更好的办法是，大声地解释给别人听。这样你会学得更快，而且可能会有一些新的认识，而这是以前光看不说的时候未曾发现的。

8. **听听你的大脑怎么说。**
   注意一下你的大脑是不是负荷太重了。如果发现自己开始浮光掠影地翻看，或者刚看的东西就忘记了，这说明你该休息一会了。达到某个临界点时，如果还是一味地向大脑里塞，这对加快学习速度根本没有帮助，甚至还可能影响正常的学习。

9. **要有点感觉！**
   你的大脑需要知道这是很重要的东西。要真正融入到书中的故事里。为书里的照片加上你自己的说明。你可能觉得一个笑话很憋脚，不太让人满意，但这总比根本无动于衷要好。

10. **等完全看完这本书之后再做最后的模拟测验（Coffee Cram Mock Exam）。**
    如果太早做这个模拟测验，你就会不清楚自己为真正的考试准备得怎么样。等你觉得准备得差不多了，再像参加最后的考试一样，在两个小时的时间内完成这个模拟测验。如果做得太早，等你下次再做的时候，这就不是一份新考卷了，你会对其中的一些题目留有印象。

# 看这本书需要些什么：

要把你的大脑开动起来，需要一支笔，除此以外，你还需要Java、J2EE和一台计算机。



其他的开发工具就不需要了，比如，你不需要有一个集成开发环境（Integrated Development Environment，IDE）。我们强烈建议，在学完这本书之前只用一个基本编辑器就可以了，其他的工具先不要用。支持bean的IDE会隐藏一些细节，你将无法切身体会这些细节，而这些内容又确实很重要（可能正是你要测试的），所以，最好完全手工地开发bean代码，而不要借助某个工具的力量。等你真正理解了到底会发生什么事情之后，可以再转而采用一个工具，让它自动完成创建bean的一些步骤。建议你使用一个工具来构建XML部署描述文件，可以使用J2EE参考实现部署工具*deploytool*或xDoclet。



## 得到J2EE

- 你需要1.3或更高版本的J2SE SDK。尽管没有明确说J2EE能支持J2SE 1.4或更高版本，但实际上支持J2SE 1.4也不成问题。不过，规范只是保证能支持J2SE 1.3。（你可以自己测试看看！）
- 如果还没有一个EJB 2.0兼容的服务器，可以先到java.sun.com下载J2EE 1.3。它包括了整个J2EE API的JAR文件（包含EJB开发所需要的类），还包括一个用于实践的参考实现（Reference Implementation，RI）服务器。
  先不要装J2EE 1.4！SCBCD考试是基于J2EE 1.3的，因为1.4还太年轻，在企业界没有得到广泛的使用。我们认为，最早到2004年底以前，不会有太多的企业系统移植到J2EE 1.4。等到J2EE 1.4的支持率大幅攀升时，这个考试也会升级为新的1.4规范。
- J2EE SDK不包括API文档，但是对你来说API文档是不可少的。再回到java.sun.com，找到并下载J2EE API文档。你也可以不下载，而是在线访问API文档，不过这样在查看文档的时候会有些费时费力。
- 下载J2EE SDK并解开后（用什么办法解开，这取决于你下载的SDK的版本，还要看你用的是什么操作系统），如果把J2EE RI作为实践所用的服务器，还需要增加一些环境变量。下载包中提供了有关的说明，不过并没有说怎么修改你的PATH和CLASSPATH，也没有说怎么增加两个新的环境变量：J2EE_HOME和JAVA_HOME。
- 如果你要参加考试，还需要拿到J2EE规范。下载EJB 2.0（记住，不是2.1！）规范，随时查看。考试内容不会超出这个规范的范围。我们提供的模拟测验答案中，大多都指出了规范中与这道题对应的是哪几页，如果你这道题做得不对，可以翻到那几页，多了解一下你为什么会做错。



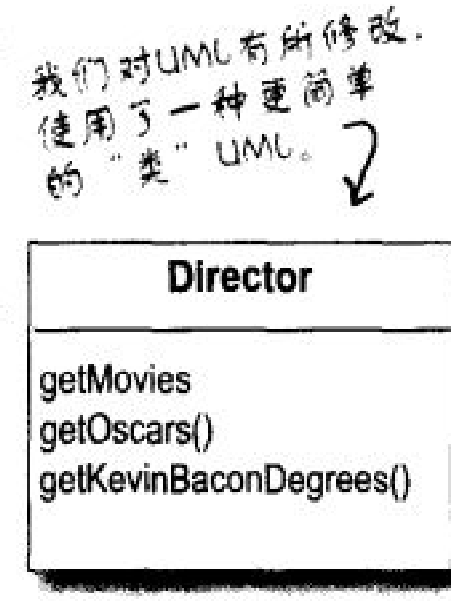
Java 2 标准版（J2SE）1.3

Java 2 企业版（J2EE）1.3
（包括EJB 2.0）

注意：这本书里的许多代码都可以从wickedlysmart.com下载。

# 需要了解的最后几点：

要把这看做是一个学习的过程，而不要简单地把它看成是一本参考书。我们在安排内容的时候有意做了一些删减，只要是对有关内容的学习有妨碍，我们都毫不留情地把这些部分一律删掉。另外，第一次看这本书的时候，要从第一页看起，因为书中后面的部分会假定你已经看过而且学会了前面的内容。




**我们用了"类"UML图（注意，可不是UML类图，而是指与UML图很相似）。**

你很可能早就会UML了，不过这个考试对UML没有要求，所以在学这本书之前不必非要先学UML。你不用担心在学EJB的同时还要学UML。

**我们没有把规范中的细枝末节一网打尽。**

SCBCD考试是很重细节的，当然我们也一样。但是，如果规范中的某个细节不在考试范围内，我们就不会谈到这个内容，除非它对大多数组件开发人员都很重要。开始开发EJB组件需要知道些什么？要想通过考试又需要知道些什么？其实这二者有95%都是重叠的。我们也谈到了一些考试不要求的内容，不过会特别指出这是考试不要求的，如果只是要通过考试，你就不必去记这些内容。另外，还有很少的一部分生僻的细节，这些内容我们没有做详细介绍，但在考试中有可能会出现。实际的SCBCD考试题就是我们出的，所以我们知道你应该把劲往哪里使！如果考试中的某个问题问到了这样一个过于"吹毛求疵"的细节，但是我们觉得学习这一点有些得不偿失，费很大的劲收获却不大，我们就会忽略这个细节，或者轻描淡写地提一下，也可能只是在一道模拟测验题中问到。

**书中的实践活动不是可有可无的。**

我们向你提供的是一个学习过程，而不完全是一本读读而已的书。这里的练习和实践活动可不是可有可无的装饰和摆设；它们也要算本书核心内容的一部分。其中有些练习和活动有助于记忆，有些能够帮助你理解，还有一些对于如何应用你所学的知识很有帮助。如果你确实想学会并记住，而且想一次就通过考试，千万不要小看或忽略任何内容。

**我们有意安排了许多重复，这些重复非常重要。**

Head First系列的书有一个与众不同的地方，这就是：我们希望你确确实实地学会，而且希望你在学完本书之后能记住学过了什么。尽管重复很有必要，不过大多数参考书都不认为重复和回顾是一个重要的环节，但是在这本书里你会看到一些概念出现了很多次。研究表明，通常至少需要重复3次，大脑才能把学习的内容真正长久地记住。

**代码示例尽可能短小精悍。**

如果查了200行代码才能找到你要理解的那两行代码，这是很打击人的。这本书里的大多数例子往往都开门见山，作为上下文的代码会尽可能的少，这样你就能一目了然地看到哪些东西是需要你学习的。别指望这些代码很健壮，要知道这里的代码甚至是不完整的。开发健壮而且完整的代码正是你学完本书以后要做的作业。书里的示例特意写得很简单，以便于你学习，这些示例的功能往往不太完备。要想得到更详细的代码示例，可以到wickedlysmart.com上看看。

# 关于SCBCD考试

### 我得先通过SCJP吗？

是的。不管是Sun认证业务组件开发人员（Sun Certified Business Component Developer，SCBCD）考试还是Sun认证Web组件开发人员（Sun Certified Web Sun Certified Component Developer，SCWCD）考试，实际上所有开发人员考试都要求你首先是一个Sun认证Java程序员，也就是必须先通过Sun认证Java程序员（Sun Certified Java Programmer，SCJP）考试。

### 考试中有多少个问题？

参加考试的时候，你要回答70个问题。每个人回答的问题可能不一样，我们提供了很多套题目。不过，每个人拿到的题目难度都一样，而且每套题都会全面地覆盖所有内容，没有特别的侧重。在实际的考试中，针对大纲中的每个考试要求，都至少会有一道题，有些考点可能还涉及不只一个问题。

### 我要在多长时间内完成考试？

考试时间是两个小时（120分钟）。对大多数人来说，这都不成问题，因为这些问题都不会很长、很复杂，不会让你太为难。大多数问题都是很简短的多项选择，所以一下子就能确定是不是知道答案。

### 题型是什么样的？

与我们最后提供的模拟测验题几乎一样，只有一个大的差别，因为题目是多项选择，实际的考试会告诉你每道题有几个正确答案，但是模拟试题没有明确指出这一点。另外，你在实际考试里可能会遇到这样一些题目，要求把什么东西拖放到某个位置，这种问题在模拟测验里没有出现。不过，这种拖放问题实际上就是用交互的方式来回答匹配问题，也就是一个东西和另外哪个东西匹配。

### 要正确地回答多少个问题才算通过？

必须答对45个问题（64%）才能通过考试。回答完所有问题之后，不要急匆匆地就按下完成（done）按钮，可以让鼠标在这个按钮上停一会，等有足够的勇气了再按。因为一旦按下，几乎立刻（大约6纳秒）就能知道你是否通过（当然，我们相信你一定能通过）。

### 为什么这本书的模拟试题不指出有多少个正确选择？

我们希望模拟试题比实际的考试要稍微难一点，这样你就能对自己是否已经做好准备有一个最真实的认识。通常，人们在做书上的模拟试题时得分总是高一些，因为这套题目可能做了不只一次，这样一来，你可能会对自己的准备程度有过高的估计，我们可不希望你在估计不足的情况下草率参加考试。

### 考试之后我能拿到什么？

离开考试中心之前，记得要拿到你的测验报告。你在每一部分得分多少会在这个测验报告中有一个总结，另外报告中还会指出你是否通过了考试。要留好这份报告！这是你通过认证的第一份证明。考试完再过几周，你会收到Sun教育服务机构（Sun Educational Services）寄出的一个小包裹，这里面有你真正的纸质证书，Sun给你的一封贺信，还有一个可爱的胸针，上面有“Sun Certified Business Component Developer（Sun认证业务组件开发人员）”几个“小字”，这几个字实在是太小了，所以就算是你凭这个胸针谎称自己得到了其他认证，可能也没人能看出来有什么分别。如果你想喝点酒庆祝自己通过了考试，别指望Sun教育服务机构会随包裹给你寄瓶酒。

### 考试花费是多少，怎么注册呢？

考试的价格不菲，需要$150。所以你很需要这本书……以保证一次就能通过考试。你要通过Sun教育服务机构注册，向他们提供你的信用卡号（这样Sun教育服务机构的人就可以凭此转账）。相应地，你会得到一个准考证号，用这个准考证号就可以与离你最近的考试中心约一个考试时间了。

要想从网上了解有关的详细情况和购买一个准考证，可以先访问http://suned.sun.com。如果你在美国，请浏览http://suned.sun.com/US/certification。如果你不在美国，可以从主页上选择你所处的国家。

### 考试软件是什么样子？

这个软件用起来极其简单，就是向你问一个问题，再由你回答。如果你还不想回答，可以先跳过去，等以后再答。如果确实回答了，但是没有把握，希望在有时间的情况下再检查一下，可以把这个问题“标出来”。做完之后，你会看到一个界面，其中显示了哪些问题你还没有回答，另外哪些问题是加了标记的，这样你就能直接返回去再完成或检查这些题目。

在考试刚开始的时候，会给你提供一个简单的教程，告诉你怎么使用这个软件，还会让你做一个小测验（与EJB无关）练练手。学习这个教程所用的时间不算在EJB考试时间内。等你学完这个考试软件教程，而且确实准备好了，时钟才开始计时。

### 到哪儿能找到一个有关这个考试的新闻组，要准备多久才能参加这个考试？

要说关于这个考试的在线新闻组，最好的当属本书作者维护的讨论组了！（当然了，这有什么奇怪的？）。你可以访问javaranch.com，然后浏览Big Moose Saloon（所有论坛都在这里）。这个讨论组很有用，可别错过了。在那里肯定会有人回答你的问题，可能正是我们自己来回答你的问题。Javaranch是互联网上最友好的Java群体，所以不管你的Java水平如何都不用害怕，这里欢迎每一个人。如果你还需要通过SCJP考试，同样可以在这里获得帮助。

要准备多久才能参加这个考试，这要取决于你有多少EJB经验了。如果你以前从未接触过EJB，可能需要3～10周的时间，具体要多久要看你每天在EJB上花多少时间。

### 为什么这个考试是基于J2EE 1.3而不是新的1.4？

Sun的各种考试的目的是为对某项技术至少有6个月经验的人提供认证。这是不是说，一旦J2EE 1.4推出6个月之后，SCBCD考试就会更新到这个新版本呢？不是这样的。因为发布1.4的时候，各个公司还没有打算把服务器移植到这个版本。只有当业界有足够多的公司转去采用1.4版本时，这个考试才会升级到J2EE 1.4。如果你到一家公司应聘，声称自己“嘿，我已经通过这个技术的认证了，而且是这个技术最新最好的那个版本，嗯，虽然我发现好像还没有谁真正使用这个版本，不过……”这么说对你没有任何帮助。Sun的考试不是在考“你知道多少”，而是要考“你能应用多少”。这些考试都是专业考试，许多问题都是精心设计的，你可能知道EJB如何工作的有关内容，而这些问题则是要测试你是否真正理解这些内容有什么含义。

# 1 EJB入门

# 欢迎来到EJB世界




**Enterprise JavaBeans（EJB）并不难。**不错，想想看，如果让你自己动手写一个可扩展性、事务性、安全性和并发性兼备的企业服务器，该是多么困难，所以至少从这一点来看，相比之下还是使用EJB简单得多。这一章我们将开发、部署并运行一个EJB应用，然后再深入讨论它的细节。在这一章的学习过程中，我们将分析怎样使用EJB，EJB有什么好处，它有哪些特点，而且我们还会简要谈到EJB容器如何工作。我们将从一个高层次来分析EJB的体系结构，并了解三种bean类型。你对这一章的内容理解得越多，以后要记的东西就越少，所以千万不要把这一章跳过去不读（如果你是一位EJB专家，可能快速浏览一遍就够了）。

# OBJECTIVES

## EJB概述

### 标准大纲：

1.1 明确EJB技术的使用、好处和特点（针对EJB规范2.0版本）。

### 具体含义：

需要知道EJB整体是怎么工作的，它有什么好处，EJB提供了些什么，以及哪些是它没有提供的。

需要理解EJB的整体架构，还要知道这个体系结构如何支持EJB的各项特性。例如，你要知道EJB能够支持事务、安全性和并发性，不过它不能保证负载平衡、故障恢复或集群。你还要知道EJB支持3种bean类型：会话bean、实体bean和消息驱动bean。另外，会话bean又可以有无状态和有状态之分。

# EJB到底是什么？

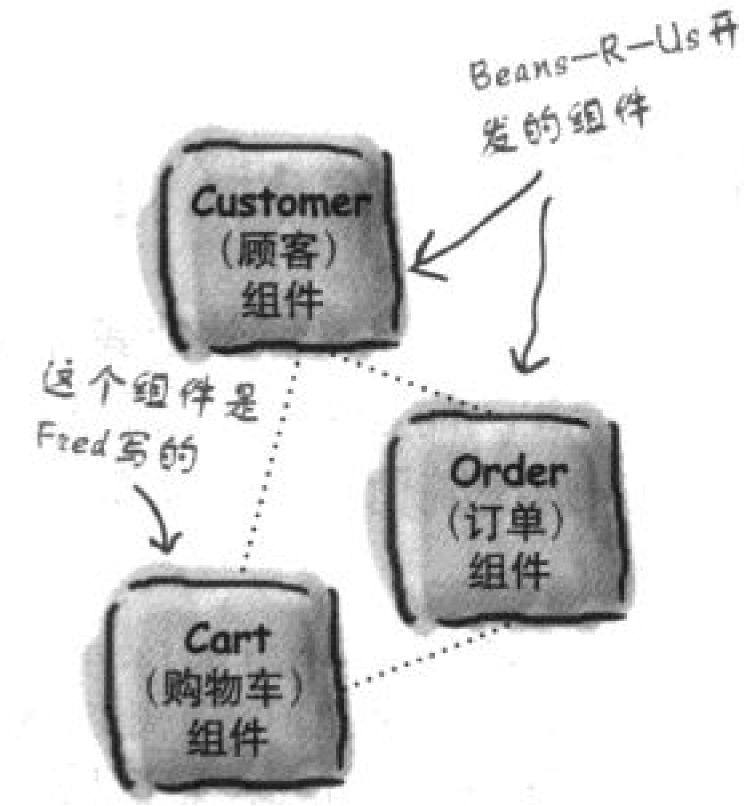
## 基于组件的开发

利用Enterprise JavaBean（EJB），你就能像搭积木一样开发EJB组件，然后你或者是其他人可以把这些EJB组件组装或重新组装到不同的应用中去。例如，你可能创建了一个Customer（顾客）bean（bean就是组件，只是叫法不同），它表示数据库中的一个顾客。这个Customer bean就能用在多个应用中，比如可以用在审计程序、电子商务购物车系统或技术支持应用中，只要应用中需要表示一个顾客，几乎都可以使用这个组件。实际上，对于某些bean来说，bean的开发人员和使用bean的应用组装人员可能不在同一家公司工作，甚至有可能彼此互不认识。

如果你是一位bean开发人员，可以构建一个Order（订单）bean、Payroll（工资）bean，或者一个ShoppingCart（购物车）bean，这样其他公司的开发人员（可能与你毫无关系）就能向你购买这些组件，并用来建立他们自己的应用。

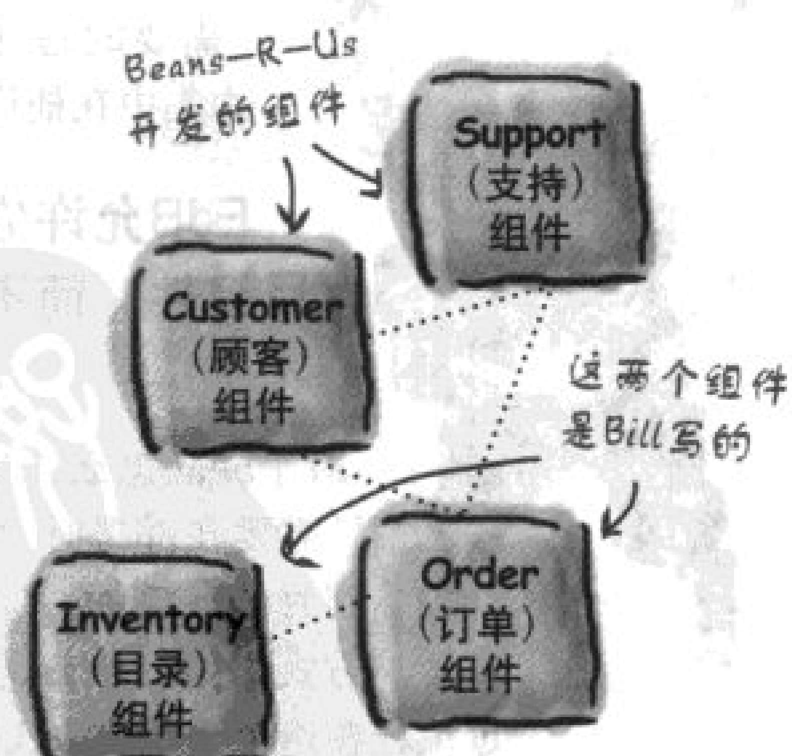
基于组件的开发有一大妙处，这就是：基于组件，可以让代码重用上升到一个新的高度。这里不是重用Java类，而是重用一个更大的功能块。bean要怎么工作通常完全可以从外部进行修改，根本不用动bean内部的Java代码！在这一章中你将了解到，要在服务器上部署一个bean，可以用声明方式配置和定制这个bean（也就是说，通过一个基于XML的部署描述文件），这样只需修改部署描述文件就能改变bean在运行时的表现。

**利用基于组件的开发，可以把代码重用上升到一个新的高度。利用面向对象开发，重用的是类，而基于组件时，重用的则是更大的功能块，不用触及具体的代码就可以定制这些可重用的组件！**




**应用A：在线购物**

Fred用3个组件组装了一个在线购物应用，这个应用里使用了两个从Beans-R-Us公司买来的组件，还用了Fred所在公司自己开发的一个组件。




**应用B：技术支持**

Bill用4个组件组装了一个技术支持应用，其中两个组件是从 Beans-R-Us买的，另外两个组件是他自己开发的。

# EJB到底能给我们什么？

**利用EJB，你就能集中精力来处理企业的业务逻辑，而像事务、网络、安全等等这些底层服务则统统留给EJB服务器开发商来解决。**

假设你为Guitar Land公司工作，这是一家在线销售音乐器材的公司。你完全不必每周工作90小时这么辛苦，那么你打算把时间都用到哪儿呢？是不是应该专心考虑Guitar Land怎么在线完成业务，而不是编写你自己的一个安全的、网络化的事务管理服务器？为什么不做你最了解、最拿手的事情（所在企业的业务逻辑），而把重活（也就是服务器提供的所有基础服务）留给别人来做呢？



EJB模型的目的是让每一个人都尽其所能，服务器开发商的重点是大多数企业应用都需要的基础架构，而业务开发人员则把精力集中在他们自己的业务逻辑上。

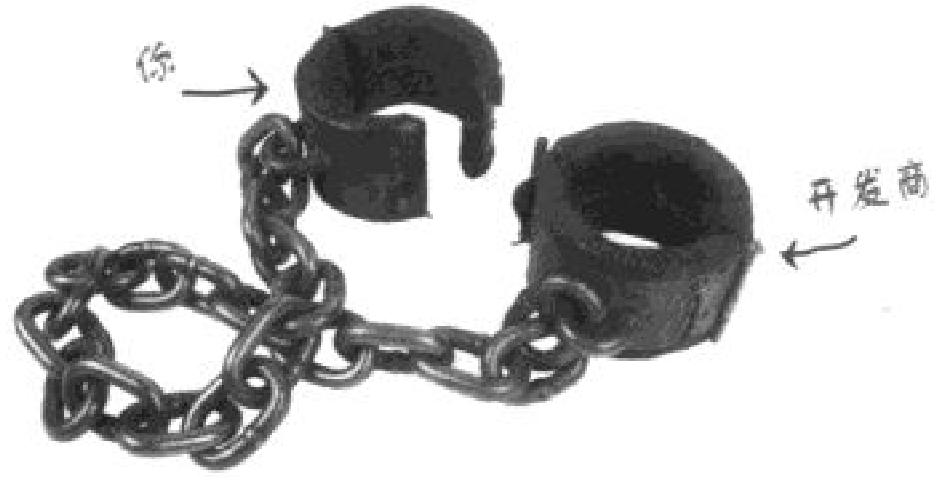
**EJB允许你在部署时定制和配置可重用组件，而不用去动它的源代码！**

对于bean怎么使用这些底层服务要想有所修改，只需要在部署时对一个XML文档稍加调整即可达到目的。例如，完全可以在XML文档中（用声明方式）定义bean方法的安全访问控制，而不必通过编程在这个bean的源代码里做工作。另外，还可以定制bean方法在事务中以何种方式运行，这些都是在部署描述文件中完成的，你不用把事务边界和行为硬编码写到程序里。这正是关键所在。

**EJB服务器已经提供了许多服务，所以你不必亲自动手来编写这些服务：**

- **事务管理**
- **安全**
- **并发**
- **网络**
- **资源管理**
- **持久存储**
- **消息传递**
- **部署时定制**

已经有人编写而且测试了以上的所有服务，所以你没有必要再自己来写。




## 不再为开发商所困！

**企业bean是可移植的，这不仅是指它们可以移植到不同的JVM中，更重要的是可以移植到不同的EJB服务器。**




我们之所以如此热爱Java，原因之一就是它能在多种平台上移植。这就是所谓的“一次编写，到处运行（write-once-run-anywhere，WORA）”。

EJB则把可移植性提升到一个新的高度，这里不只是“一次编写，到处运行”，还能做到“一次编写，到处部署（write-once-deploy-anywhere，WODA）”。WORA可以让你摆脱运行平台的限制，这样就能让你的应用不只是在一种操作系统（OS）上运行；相比之下，WODA则更进一步，有了它你就不用再看应用服务器开发商的眼色了。以后没准还有什么“YODA”，也就是说，不用编写，只需把某个工作做一次就能到处部署，不过这有些离题了，我们还是言归正传吧。

在过去，每个应用服务器开发商都有自己的专用API。我们不得不去学、去用这样一个专用API，最后才能让我们的企业应用启动并运行起来。想想看，接下来会怎么样？你发现还需要一个新特性。再然后又如何呢？你的开发商可能会说：“我们正在考虑在下一版里增加这个特性……这个版本可能在明年才会推出。”现在该怎么办呢？就像一个吸毒上瘾的人一样，你在不知不觉中已经上了开发商的套，如果现在考虑放弃，实在让人很难承受。放弃这个开发商的应用服务器又能怎么样呢？接下来还不是要投靠另一个开发商，去学另外一套专用API，这不又被困住了吗？

EJB的一个最重要的优点就是WODA。现在开发商不仅要在销售上竞争，也就是不仅要能把产品卖给你，还要能留住你。因为每个人都知道，你完全可以收拾好你的bean，转而去找其他的开发商。

## there are no Dumb Questions

**问：** EJB与J2EE有什么区别？

**答：** J2EE和EJB都是服务器的规范。不过，EJB实际上是J2EE的一个子集。换句话说，J2EE服务器必须包括一个EJB容器。前面我们已经用过"服务器（server）"这个词，但从技术上讲，企业bean（enterprise javabean）实际上是在一个称为EJB容器（EJB container）的环境中运行的。所以，每个J2EE服务器都必须包括一个EJB容器，当然除此之外J2EE服务器中还要有一个能运行servlet和JSP的Web容器。

这个考试（SCBCD）针对的是J2EE的EJB部分，SCWCD考试（Sun Certified Web Component Developer，Sun认证Web组件开发人员考试）针对的是J2EE的Web部分。

在这一章的后面，我们会更详细地介绍J2EE和EJB有着怎样的关系，二者如何结合。

在本书中接下来的内容中，我们会交替地使用"EJB服务器"、"容器"和（单独的）"服务器"这几个词，它们指的都是EJB容器。如果在某种情况下这些词确实有所区别，我们会特别指出来。

**问：** 能不能脱离EJB兼容的应用服务器来使用EJB组件？

**答：** 绝对不行。EJB组件是无法在EJB容器之外"存活"的。EJB组件没有主方法，即便你刻意地向你的bean类增加这样一个主方法，这个bean就其本身来说也没多大用处。企业bean里的大多数方法都是由容器本身调用的，如果脱离了服务器，这些方法就没有任何意义了。要记住，EJB服务器最重要的一点就是它能为你提供所有重要服务（安全、事务等等），如果没有服务器，除了你的基本业务逻辑外，你将一无所有。而且，如果你的业务逻辑要依赖于容器（比如，要调用容器所提供接口上的方法），那么就算这只是你的业务逻辑，在没有服务器的情况下也将遭遇失败。

**问：** 我能这样设计和编写我的代码吗？就是把大部分业务逻辑都放在一个无格式的纯Java类里，然后只是让bean调用这个类的方法，这样行不行？这样的话，我就还可以重用这些业务逻辑了……

**答：** 可以，这样做是可行的。实际上，很多设计人员都采用了这种做法，先编写单独的可重用类，这些Java类不是bean，然后再让bean调用这些类的方法。如果你的bean调用了一个Java类（不是bean）的方法，这个方法仍然在容器的控制之下，所以对容器而言，这个非bean方法就是这个bean的功能的一部分。

**问：** 常规的javabean和企业javabean有什么区别？

**答：** 祝贺你！你是第3 000 000个问这个问题的人了。

"javabean"这个词的意思是可重用组件。常规的非企业bean（bean只是javabean的一种简写形式）就是遵循一种命名约定的可重用组件，这些常规的javabean可供开发工具使用。

到目前为止，最常用的javabean要算GUI组件了，比如说Swing按钮或文本框之类。几乎所有Java IDE（集成开发环境）都是javabean兼容的，所以如果你在使用一个可视化的布局工具，可以点击一个按钮，然后会弹出一个属性表，在这个属性表里可以设置按钮的颜色、大小、字体等等属性。布局工具知道这个按钮bean有哪些属性，因为bean肯定遵循一组命名约定，提供了获取方法（getter）和设置方法（setter）。

不过，常规的javabean不只是用作GUI组件，其他Java技术也可以使用javabean特性，这包括Jini和Servlet等等。

企业javabean也是可重用的组件，不过它与常规javabean的相同之处仅限于此。常规javabean中的"bean"主要是指在开发时可重用，这样在建立一个bean的事件与另一个bean的方法之间的关联时能够有所简化或加快速度，或者可以更容易地设置性质值（通常这与实例变量值含义相同）。常规的bean在JVM中运行，就像所有正常的Java类一样。与此不同，企业bean中的"bean"则是指在运行时可重用，而且企业bean必须在EJB容器的控制之下运行。

**在这本书余下的部分中，当我们谈到bean时，指的就是企业bean。**

# 它是怎么工作的？

**你的bean要在EJB服务器的控制（和保护）之下运行。**

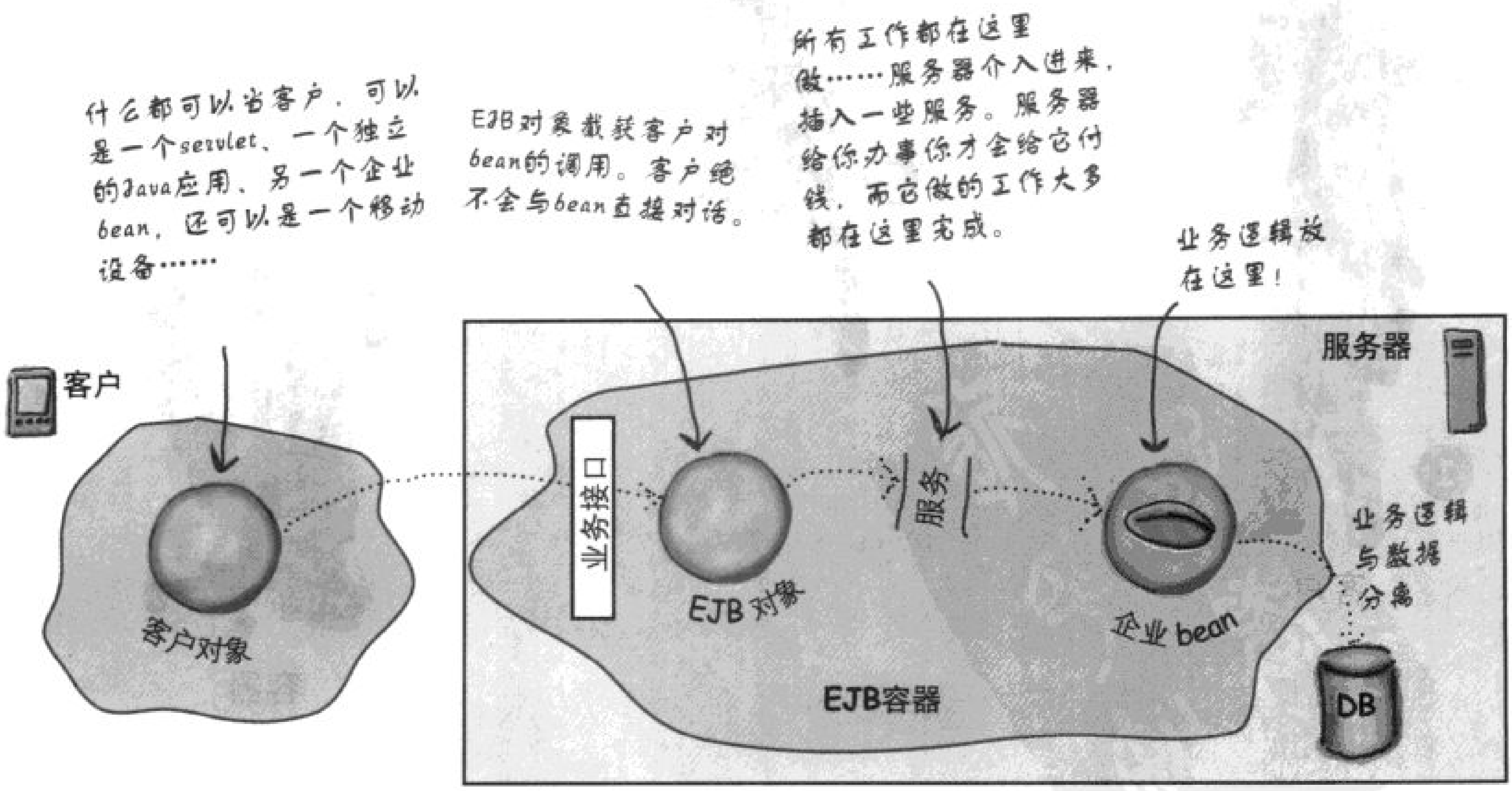
**每次客户向bean发出方法调用时，服务器都会介入到客户和bean之间，插入一些“服务”，如安全、事务和持久存储等等。**

你的bean是在服务器中“生存”和运行的。服务器身负重任，它要竭尽全力去管理事务、安全和持久存储，甚至对象的生和死也在服务器的管控之下。所有这一切都是通过插手做到的，每次客户做一个请求时（也就是说，调用bean上的一个业务方法），服务器就会插手。此时服务器会介入进来，问这样一些问题：

“这个客户有足够的安全性来调用这个方法吗？”或者

“这个bean要作为一个更大事务的一部分来运行吗？”或者

“这个bean为客户运行那个方法之前，需不需要用数据库里的数据把自己刷新一下？”




EJB体系结构的高层视图（漫画版）

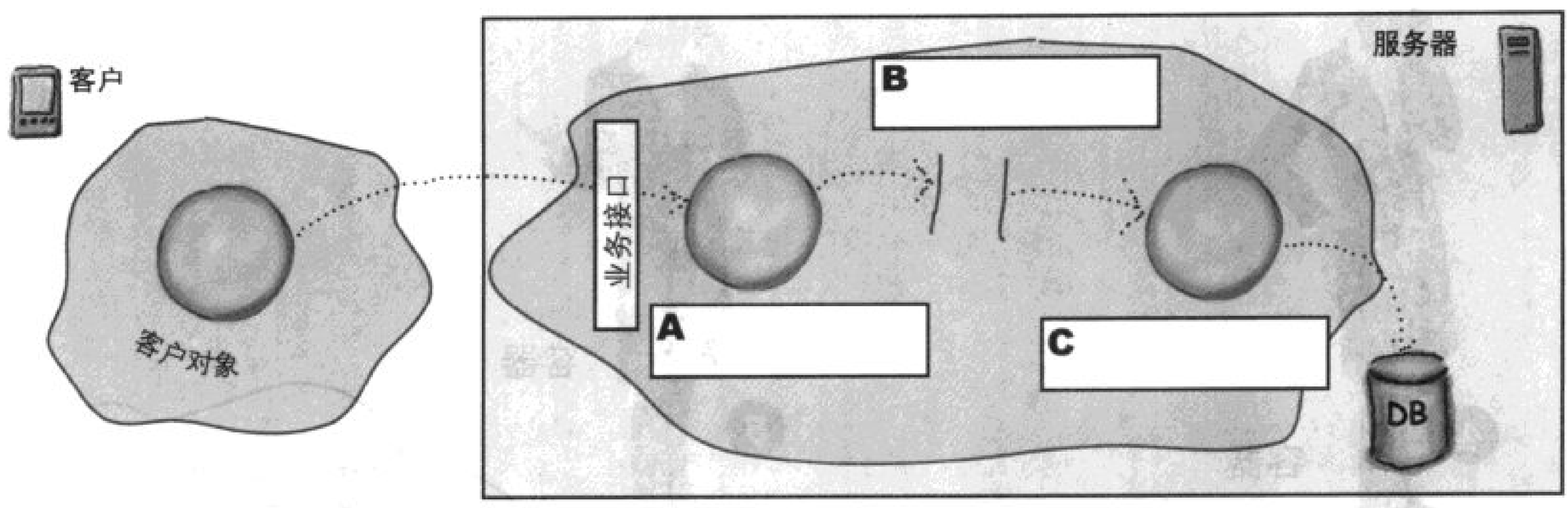
# 幕后的动作……


嗯，对，请Broker bean接电话，我现在要和他谈笔生意。
1
客户
别那么急匆匆的，伙计。除了容器，其他的人谁也不能和bean直接对话，任何人都不行。如果你真想和bean联系，首先要通过我这一关。给我看看你的ID，我会向容器核对，如果没问题，我会负责转告你的请求。
2
EJBObject
让我看看……看来在批准名单上确实有他，他应该可以对Broker bean调用Trade()。
3
容器


我要请一位Broker bean现在下来。Slim Sleazy想做笔生意。噢，对了，bean下来的时候别忘了带上自己的新事务。
4
容器
什么？你说什么？没有bean了？从freakin池里找一个不就得了！没关系，哪一个无所谓。是，我知道这些bean都被惯坏了，喜欢耍点大牌，不过，现在确实需要他们上场了。
5
容器
嘿，上一次 可是我去的，所以这一次该轮到你们谁去了。再说了，我还没从上一次那个大事务回滚中恢复过来呢。
他们给我们的太少了！要知道我们才是真正干实事的，业务逻辑全由我们负责。不过，好在至少我们不用和客户直接打交道。不错，有一个EJBObject做保镖真 是 很棒。
6
bean池
好的，bean准备好了，那就继续，把客户的方法请求转告给这个bean。还有，顺便说一句，请那位“bean皇后”小姐最好穿戴整齐一点。我已经对她进行了实例化，我可以把她送到垃圾回收站去。
7
容器

## Sharpen your pencil

① 填空，写出图中三个部分分别是什么。




② 简要描述这三个部分分别负责什么，或者说明它们如何工作。

A ______________________________

______________________________

B ______________________________

______________________________

C ______________________________

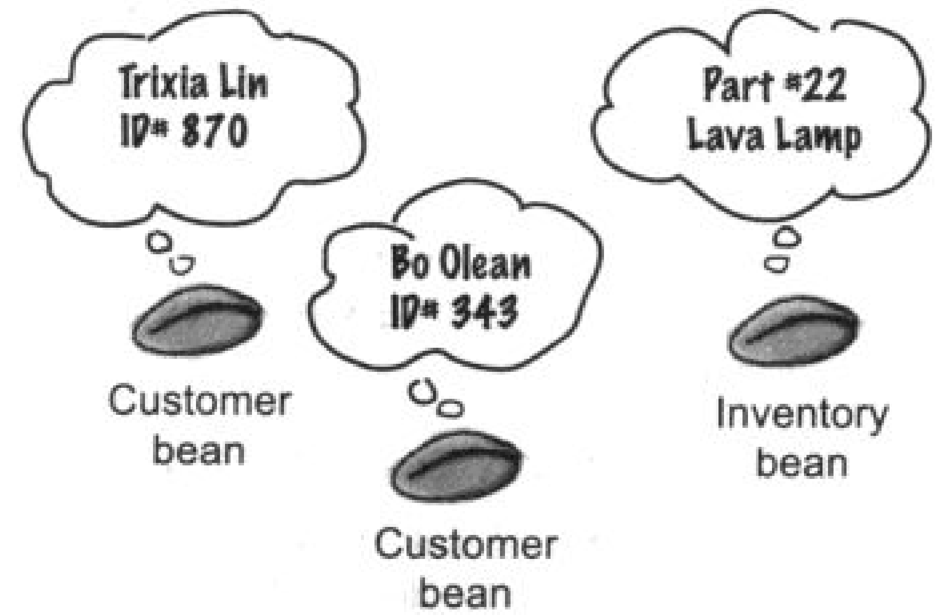
______________________________

# bean的三种类型

## 实体bean（entity bean）



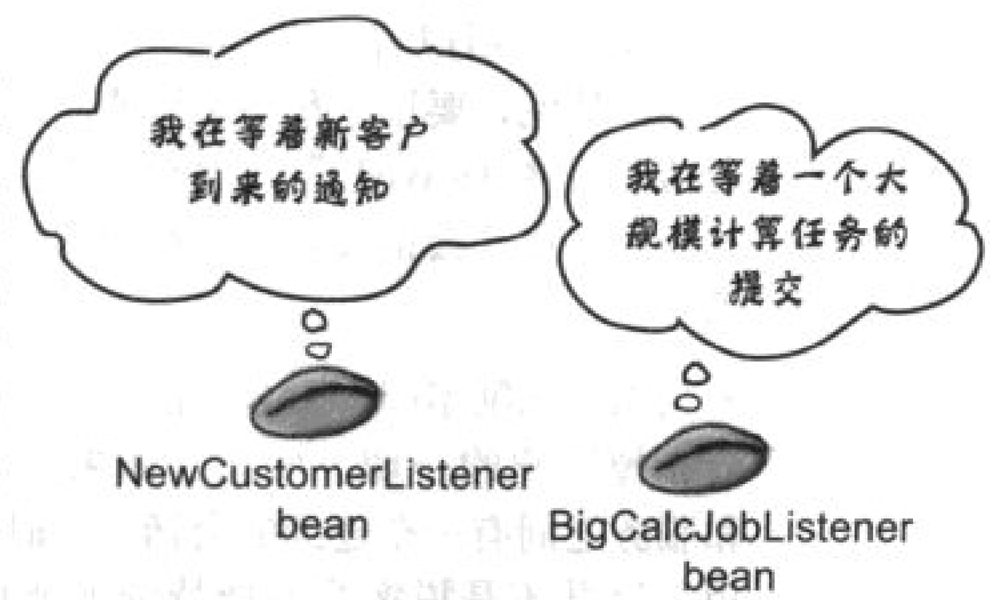

实体bean用来表示持久存储库里的一个东西。这通常就是指数据库里的一个东西，实体bean的一个实例就表示数据库表中的一行（不过，如果数据库得到了规范化，bean也有可能从多个表的数据行提取信息，也就是说，可能对应到多个表中的多个数据行）。实体bean的一个典型例子就是Customer（顾客），一个Customer实体bean可能表示Bo Olean（ID# 343），另一个Customer实体bean可能表示的是Trixia Lin（ID# 870）。




## 消息驱动bean（message-driven bean）



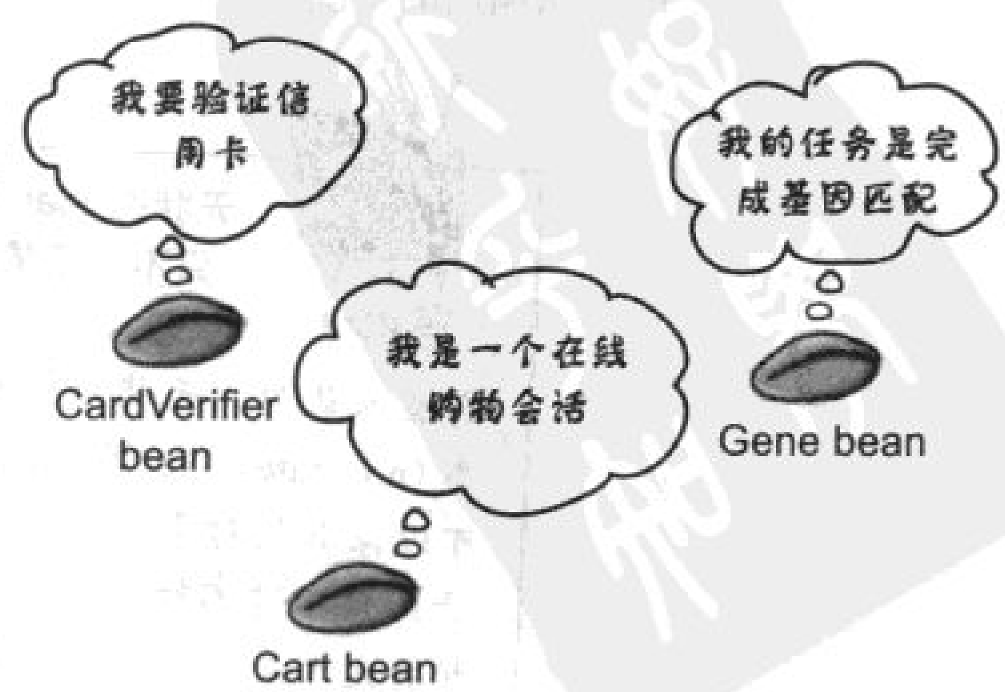
只有在需要一个JMS客户的时候才会使用消息驱动bean。换句话说，消息驱动bean可以监听来自JMS消息服务的消息。客户绝对不会直接调用消息驱动bean；要想让消息驱动做点什么事情，客户必须向一个消息服务发出一条消息。这说明，消息驱动bean没有相应的EJBObject，因为服务器会从消息服务直接得到客户请求，而不是当客户向bean发出调用时截获客户请求。NewCustomerNotification订购者就是一个典型的消息驱动bean。




## 会话bean（session bean）



什么时候用会话bean呢？……应该说剩下的所有事情都要用会话bean来完成。几乎各种后端服务都可以（而且通常应该）使用会话bean来实现。实体bean表示的是一个东西，而会话bean通常表示一个过程。可以换种说法来解释，当你想到实体bean的时候，应该能想到这是一个名词，而在考虑会话bean时，脑海里应该有动词的概念。购物就是一个典型的会话bean例子，信用卡处理系统也可以算作一个会话bean。



# 会话bean可以无状态，也可以有状态

在会话bean一章中我们还要详细地讨论这个内容。不过，对于现在来说，必须知道，会话bean可以（在部署时）标记为无状态（stateless）或者有状态（stateful）。

有状态bean可以记住方法调用之间的会话状态，而无状态bean对于方法调用之间的客户情况则是什么也记不住。

“会话状态（conversationalstate）”这个词实际上是指“客户特定的状态”，典型的例子就是购物车。假如你去买东西，推上一个购物车，放一件商品进去，然后再放第二个商品的时候，你发现头一个商品不翼而飞了，如果是这样，你肯定不会乐意。这对用户实在是不太友好。所以好的购物车应该保留客户的购物状态（也就是说，要记住购物车里的商品），而且只要购物会话“活着（alive）”，这个状态信息就应该一直保留（我们会在“会话bean”一章中解释“活着”是什么意思）。

无状态bean则不同，一旦方法调用结束，无状态bean就会忘掉客户的一切。所以，如果一个服务不需要客户和服务之间有一个连续的会话，此时无状态bean就很适用。这并不是说客户不能持续地调用无状态bean的方法，我们的意思是，客户不能指望这个无状态bean记住以前的方法调用的情况。

**当心！**

**无状态bean可以有状态！（只不过不是客户特定的状态。）**

有些人认为“无状态”就是“没有状态（no state）”。其实，无状态bean也可以像其他对象一样有自己的实例变量；只不过不能用这些实例变量来维护特定于某个客户的值而已。

### there are no Dumb Questions

**问：** **我听说只有无状态会话bean才能扩展，而且大家都不应该使用有状态会话bean。这种说法对吗？**

**答：** 不是这样的，这种说法不完全正确。确实，无状态会话bean一般比有状态会话bean更可扩展，这得益于容器就是以一种可扩展的方式来管理无状态bean的。在“会话bean”一章我们还会再做解释。

但是……这并不是说绝对不要使用有状态会话bean。在需要会话状态的时候就要考虑有状态bean了，且如果用其他方法保存状态（比如使用客户来保存状态，或者使用一个servlet保存状态，再或者使用一个数据库来存储客户发出各个方法调用之间的状态），但是这些方法对性能会有较大影响，相比之下，尽管有状态会话bean在可扩展性方面稍逊一筹，但在性能上更能满足要求，此时也可以考虑使用有状态会话bean。




## Sharpen your pencil

**了解bean的类型。**

先看左边的问题描述，看看哪些类型的bean适用于这些问题，对合适的bean类型打勾。需要说明，这些问题并没有一个标准答案……你可能发现用一种方法解决某个问题时，采用某一种bean类型比较合适，而用另一种完全不同的方法解决这个问题时，可能另一种bean更适用。

| | 实体bean | 消息驱动bean | 会话bean（圈出是无状态、有状态，还是二者都适用） |
|---|---|---|---|
| 预订一张摇滚音乐会的门票 | ☐ | ☐ | ☐ 有状态 无状态 |
| 银行审计 | ☐ | ☐ | ☐ 有状态 无状态 |
| 搜索一个产品数据库 | ☐ | ☐ | ☐ 有状态 无状态 |
| 配对服务完成配对 | ☐ | ☐ | ☐ 有状态 无状态 |
| 接收提交来的成本报告，再把这些报告发出，来获得批复 | ☐ | ☐ | ☐ 有状态 无状态 |
| 在线专家诊断——你来描述症状，由系统帮助你找出病因 | ☐ | ☐ | ☐ 有状态 无状态 |
| 图书馆里的书 | ☐ | ☐ | ☐ 有状态 无状态 |




**会话bean**：干这么多活，却一句夸奖的话都没有，真是受够了。

**酒吧招待**："一句夸奖的话都没有？"你是什么意思？要知道，从最早的EJB 1.0开始，规范惟一一直支持的bean不就是你吗？

**会话bean**：但是，对我来说这还远远不够。现如今，每个人都只是在谈实体bean。实体bean，实体bean，到处都在说实体bean。并不是我不喜欢实体bean，其实我的一些最好的朋友就是实体bean，不过我真的希望人们能看到我做了些什么。

**酒吧招待**：说到这里，也真的是这样，现在我们酒吧里人们最常谈起的确实是实体bean，特别是EJB 2.0中得到CMP极大提升的那些实体bean。

**会话bean**：那是另一码事……CMP做了什么？不就是和数据库打交道嘛！说正经的，你说说那有什么特别的？"嚯，看哪！它更新了一条记录！"拜托，别那么大惊小怪好不好。

**酒吧招待**：是这样，不过现在的程序员好像根本不想自己写数据库代码，所以有人能帮着做这些事情，在他们眼里当然很吃得开了。再说了，好像还有一个什么有关持久性关系的东西，我想不起来是什么了……

**会话bean**：CMR，不就是容器托管关系（container-managed relationship）吗？不错，我也承认CMR确实让开发人员日子好过多了。不过，我烦的不是这个——我知道，每个人都喜欢实体bean，但是有人想过我吗？实体bean表示的是系统中的东西，不过，要是没有我，这些东西又能干多大点事情呢！可能实体有一些获取方法和设置方法，还可能有一些查询，但业务逻辑并不多呀。要在应用中使用实体bean，就必须使用会话bean来完成业务处理。比如说，一个实体bean可能表示你现在卖的酒，还有些实体bean表示各个顾客，但如果没有你们这些酒吧招待，只是有酒和顾客又有什么意义呢？需要有人用一种有意义的方式把这些实体（酒和顾客）放在一起才行！这正是我们会话bean做的事情。我们干的是实事。是我们替客户做事情，实体bean又做了些什么呢？他们只不过是坐在那里等着会话bean用他们而已。嘿，能再来一杯吗？我还没开始说消息驱动bean呢……

[未完待续]

## 要点

- EJB是一个基于组件的开发模型。
- 组件是可重用的功能块，可以针对不同应用修改这些组件，而无需改动Java源代码。
- EJB的一个好处是WODA。可以把你的EJB 2.0组件部署到任何EJB 2.0兼容服务器上。
- WODA意味着你只需要学一种标准API，而不用去学多种特定于开发商的专用API。
- EJB体系结构使用一个EJBObject来截获客户对bean的调用。这样，服务器/容器就有机会插手来增加服务。
- EJB服务包括事务、安全、资源管理、网络和持久存储。
- bean有三种类型：实体bean、会话bean和消息驱动bean。
- 实体bean表示持久存储库中的一个可惟一标识的东西，通常表示数据库表中的一行。
- 消息驱动bean就是JMS消息服务的消费者。
- 会话bean是什么呢？剩下的都是会话bean。
- 会话bean可以有状态，也可以无状态。
- 有状态bean可以记住一个客户的"会话状态"，无状态bean则不能。

## 例子：顾问bean

深入到EJB内部之前，先来看看怎么从头开始开发、部署和测试一个bean。如果你对EJB还不太熟悉，可能现在还不能完全理解这里介绍的所有内容。不过也不用为此担心；在后面的各章中我们自然会把这些内容详细地一一道来。这个例子只是想让你对怎么开发bean以及怎么让它运行起来有一个大概的认识。

我们开发的第一个bean要用来提供顾问（Advice Guy）服务，这是一个远程服务，每次客户发出请求时，这个服务就会回应一个字符串作为建议。后面的几页将说明这个过程是怎么样的，然后我们才会真正开发这个bean，你可以把这个bean的开发、部署和运行当作一个初步教程。

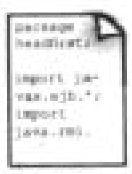



提建议的顾问*

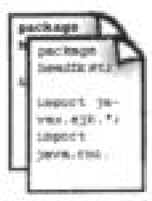
我们开发的第一个bean要用来提供顾问服务。每次客户发出请求时，这个顾问服务（这是一个企业javabean）就会回应一条很有帮助（而且相当中肯）的建议。

*在《Head First Java》中我们是用直接的TCP socket来部署这个顾问服务的。现在，只需用"区区"5倍的代码和努力就可以利用EJB来部署这个服务了。当然，有人会有另外的想法，可能会说这个顾问（Advice Guy）并不需要所有这些EJB服务，不过我们可不这样认为。虽然它现在还很"小儿科"，但我们已经在为它做IPO计划了。

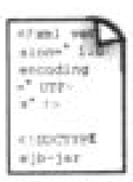
# 构建bean要做的5件事：


AdviceBean.java

❶ 编写提供所有业务方法的bean类。


AdviceHome.java
Advice.java

❷ 为bean编写两个**接口**，也就是：home接口和组件接口。

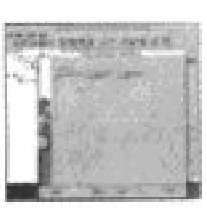

ejb-jar.xml

❸ 创建一个XML**部署描述文件**，告诉服务器你的bean是什么，它该怎么管理。这个部署描述文件的名字必须是ejb-jar.xml。



ejb-jar

❹ 把bean、接口和部署描述文件放到一个**ejb-jar**文件里（ejb-jar里面可能有不只一个bean，不过部署描述文件总是只有一个）。



❺ 用服务器开发商提供的工具把bean**部署**到服务器上。







**你不必是一个XML专家。**

实际上，你根本无需了解XML是怎么工作的。确实，有必要知道一些部署描述文件标记，但如果只是为了学这些标记而深入了解XML就没有必要了。可以把这些标记看作是文档中的一些简单标签，而且对于在这些标签里能放些什么东西有一些特殊的要求，这样就很简单了，要通过考试，你只要记住最重要的一些标签（标记）的名字和要求就行了。这些最关键的标记将在后面的一些章节中介绍。

1 **bean类**
2 接口
3 XML部署描述文件
4 ejb-jar
5 部署



AdviceBean.java

## ❶ 编写bean类，提供客户能调用的具体业务方法。

所有具体工作都在这里完成。对应组件接口中定义的业务方法，这里是这些业务方法的具体实现。换句话说，要在bean类里编写业务逻辑。

有三种bean可供选择：会话bean、实体bean和消息驱动bean，我们会在后面的几章中详细地分析这三种bean。不过，开发一个bean之前，必须先确定你需要的到底是哪一种类型，因为你的bean类必须实现一种接口，而三种不同类型的bean对应三种不同的接口，你必须根据所选择的bean类型来确定该实现哪一个接口。

这里我们打算开发一个会话bean，因为对于这个顾问应用来说，会话bean是再合适不过的了。调用getAdvice()方法时（这个方法的名字有点奇怪），顾问会回应一条建议（一个字符串）。所以我们的bean类（见下一页）实现了SessionBean接口。SessionBean可不只是一个标记接口*——这个接口里确实有你的bean类必须实现的方法。

SessionBean接口中需要实现的方法称为容器回调方法（container-callback），因为容器要用这些方法来通知你在bean的一生中有哪些重要的事件。

这个AdviceBean只有一个业务方法。

```
public String getAdvice() {
   // 生成建议的代码
}
```



**当心!**

**你必须知道怎么自己编写所有这些代码，而不要依靠一个支持EJB的开发工具来帮助你。**

你要知道怎么编写这两个接口和bean类（如果是消息驱动bean，只需要知道怎么编写bean类）。这意味着你先不要用支持EJB的开发工具，不要靠让它为你构建一些bean代码或接口代码，只有当你确实很清楚工具能为你创建哪些代码时，才可以拿来使用。

要想通过考试，有一大堆规则需要你了解，在后面的几章里我们会详细介绍这些规则。不过对现在来说，先不用操心去记这个练习中的代码反映出什么。你只要知道，这些都是需要你学会的（虽然现在我们还没有讲完，但你要知道要学些什么）。

不过，也不总是坏消息，还有一个好消息告诉你，构建XML部署描述文件是可以让工具来帮忙的。这个考试不要求你记住所有XML标记！（不过，有些元素还是要记住的；我们会在后面介绍这些重要的标记。）

*标记接口（marker interface）也称为tag interface，这种接口里没有要实现的方法，它的存在只有一个作用，有了它，你就能对外宣布："不错，这个我做得到。"



AdviceBean.java

## bean类

AdviceBean实现了SessionBean接口，所以它必须实现javax.ejb.SessionBean中声明的方法。稍后我们会仔细说明每一处细节，不过现在你只需要记住一点：具体的业务逻辑都放在bean类中。换句话说，这也是这个bean之所以存在的原因，正是为了实现这些业务逻辑，才有必要建立这个bean。对于这个顾问，要实现的业务逻辑就是getAdvice()方法。

```
package headfirst;

import javax.ejb.*;

public class AdviceBean implements SessionBean {

    private String[] adviceStrings = {  "One word: inappropriate.", "You
might want to rethink that haircut.", "Your boss will respect you if you
tell him what you REALLY think of him.", "Visualize yourself with better
clothes.", "Of course you donít have to go to work today.", "Do you really
think you should be leaving the house like that?", "Read a book, once a year
whether you need to or not."};

    public void ejbActivate() {
       System.out.println("ejb activate");
    }

    public void ejbPassivate() {
       System.out.println("ejb passivate");
    }

    public void ejbRemove() {
       System.out.println("ejb remove");
    }

    public void setSessionContext(SessionContext ctx) {
       System.out.println("session context");
    }

    public String getAdvice() {
       System.out.println("in get advice");
       int random = (int) (Math.random() * adviceStrings.length);
       return adviceStrings[random];
    }

    public void ejbCreate() {
       System.out.println("in ejb create");
    }
}
```

这个包是你需要的

你必须实现三种bean类型接口中的一种（会话bean、实体bean或消息驱动bean）

业务方法（getAdvice()）会随机地选择其中的一个串返回。

这4个方法来自SessionBean接口，所以必须把它们放在这里。对于这样一个简单的bean，这些方法不需要做任何事情，不过我们还是在这些方法里打印了一些语句，这样你就能知道什么时候调用了这些方法（或者是否调用了这些方法）。对现在来说，先不要操心这些方法是干什么的！

终于到这了！具体的业务逻辑来自组件接口。这正是这个bean的全部意义所在……这就是客户想要调用的方法。

必须有一个ejbCreate()方法。这是一条EJB原则，你后面就会学到。不过，这个方法并<u>不</u>是来自SessionBean接口。

1 bean
**2 接口**
3 XML部署描述文件
4 ejb-jar
5 部署



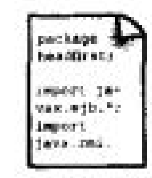
Advice.java　AdviceHome.java

## ❷ 为bean编写两个接口。

客户看到的接口有两个。我们会专门用一章来介绍这些接口，所以就目前来说，这些内容并不要求你完全理解。

### 组件接口

所有业务方法都要在这里声明。换句话说，你要把客户想调用的方法都放在这里。



Advice.java

**组件接口：业务方法**

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;

public interface Advice extends EJBObject {

   public String getAdvice() throws RemoteException;

}
```

这两条语句很重要，必须有。

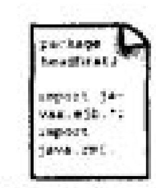
必须扩展EJBObject接口或者EJBLocalObject接口，这个内容将在后面详细说明。

这个接口中的所有方法都必须声明抛出RemoteException异常！

这是具体的业务方法。（这个bean之所以存在就是因为要提供这个业务方法）。在bean类里必须有一个与之对应的方法。

### HOME接口

客户使用home接口来请求组件接口的一个引用。home（家）可以作为客户的起点，客户从这里拿到bean的一个引用（或者至少客户认为这是bean的一个引用，不过我们会在后面说明真实情况是什么）。现在可以先把home看作是一种工厂，它能制造bean引用，而且能向客户分配bean引用。



AdviceHome.java

**home 接口：这是一个bean引用工厂**

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;

public interface AdviceHome extends EJBHome {

   public Advice create() throws CreateException, RemoteException;

}
```

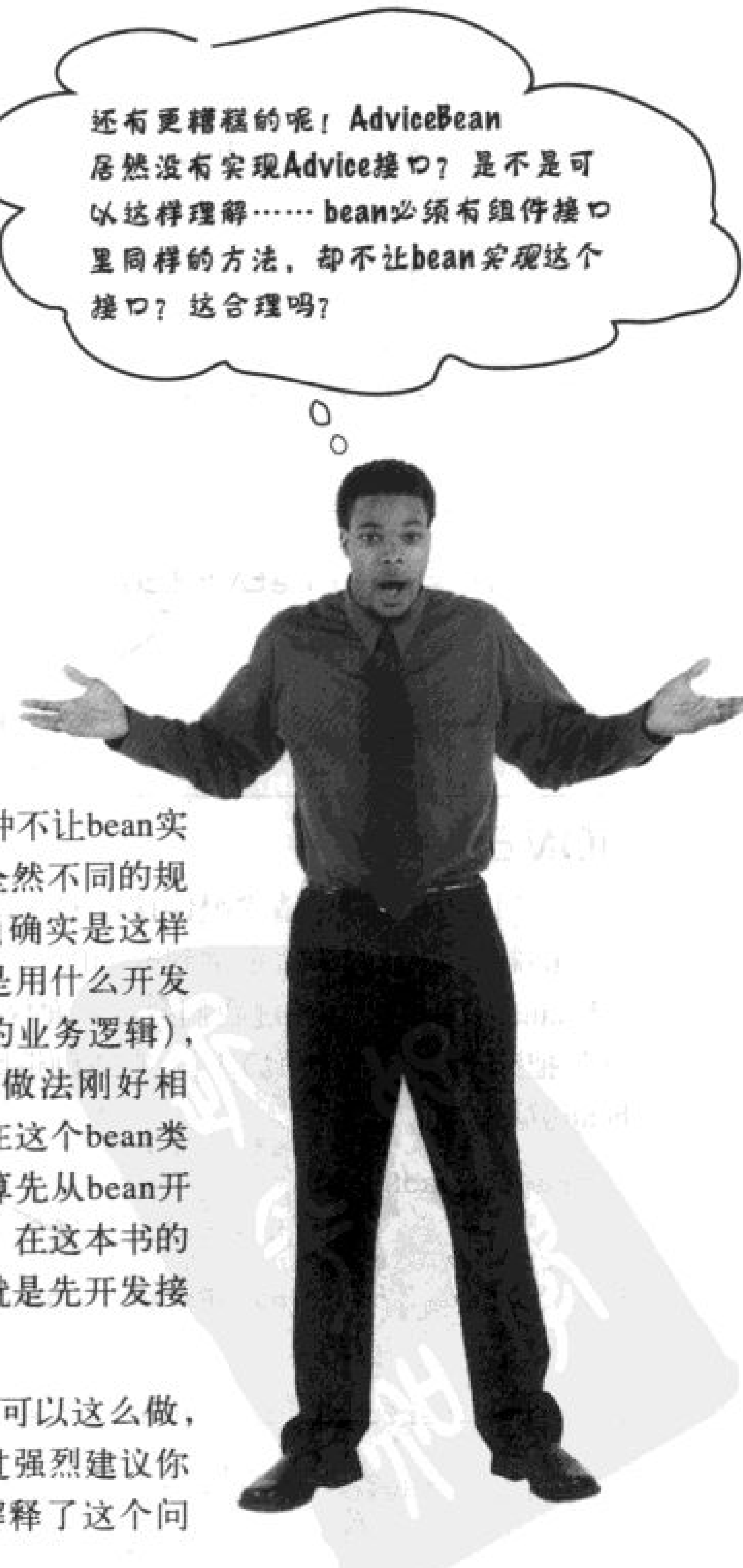
和前面一样，这两条语句很重要。

home必须扩展EJBHome接口或者EJBLocalHome接口，有关内容将在后面详细说明。

这一次需要声明两个异常：CreateException和RemoteException。

create() 方法必须返回组件接口的类型！！




嗯…… 这是两个很好的问题。在非bean的Java世界中，这种不让bean实现组件接口的做法是不太合理。不过，bean的世界有自己全然不同的规则和做法。我们当然也可以先写接口，而且有些开发人员确实是这样做的。到底先开发bean还是先开发接口，有时这取决于你是用什么开发工具来构建bean。有些工具希望你先构建bean（编写具体的业务逻辑），然后这个工具会构建与bean对应的接口。但有些工具的做法刚好相反，它会查看接口，构建一个“放具体代码”的bean类，在这个bean类中实现接口里的所有方法。为了更好地学习EJB，我们打算先从bean开始，这样可以把重点放在业务逻辑上，然后再来分析接口。在这本书的后面，我们会采用另外的一种方法来完成bean的构建，也就是先开发接口，再编写bean。

至于第二个问题，这里bean没有实现组件接口，你当然也可以这么做，也就是说，从语法上讲bean实现组件接口也是允许的，不过强烈建议你也不要这么做。请看后面的一页，在那里我们更详细地解释了这个问题。

## there are no Dumb Questions

**问：** **既然bean类（AdviceBean）要实现组件接口（Advice）中同样的方法，为什么它不实现组件接口呢？Java类可以实现多个接口，所以如果像下面这样声明应该也没什么问题吧：**

```
class AdviceBean implements
Advice,SessionBean
```

**答：** 不错，bean类是可以实现组件接口，但是规范并不鼓励这样做。要记住，尽管对客户来说，客户好像是在调用AdviceBean对象（也就是Java意义上实现组件接口的对象）的方法，但实际上并非如此，客户调用的其实是一个称为EJBObject的对象的方法，这个对象是部署时服务器实现的。客户绝对不会与bean直接交互。从来不会，而且永远不会。在这本书的后面，你会看到，如果一个bean确实实现了组件接口，可能会让一些问题偷偷潜伏起来，躲过编译器，到运行时才暴露出来，造成严重破坏。所以强烈建议你，不要让bean实现组件接口。

不过，除了上面这个问题外，这里还存在另外一个问题，组件接口扩展了另一个接口。在我们这个例子中，Advice扩展了EJBObject，而EJBObject不是一个标记接口。它本身是有方法的！这说明，任何实现了Advice的类还必须实现EJBObject的方法！

所以，你的bean最后就要实现一堆看来好像根本没有的方法，比如getHandle()、getEJBHome()……。

**问：** **不过，对这个问题还有一个很容易的解决办法呀，可以让bean扩展这样一个类，由这个类提供编译器所需的所有实现（方法实现），这样就不用在你的代码中具体实现这些方法了。这就像AWT里的适配器事件监听者类一样。这里为什么不这样做，为bean建立一个超类来实现这些方法呢？**

**答：** 是的，可以这样做，这么做确实也合法。不过，这还是意味着bean要提供一些可以调用的方法，而bean本身对这些方法一无所知。但是，EJBObject的方法是什么呢？它们是客户在调用bean时实际调用的方法，而不是bean具体要实现的方法。所以，这不算是一个面向对象最佳实践。

说bean实现组件接口不是一个好的做法还有另外一个原因，如果接口是远程的（EJBObject就是远程的，因为它扩展了java.rmi.Remote接口），这就会让实现这个接口的bean类成为一个Remote类，而这是绝对不应该的！bean在服务器的保护之下，绝对不能通过其他方法来访问，除了服务器谁访问它都不行。只有服务器才能建立EJBObject（通过实现Advice接口），EJBObject才是远程的，它会截获客户对bean的所有业务方法调用。

**问：** **但是如果不让bean实现组件接口，也就是说，如果 AdviceBean没有实现Advice接口，这就意味着倘若bean没有遵从接口，接口里有的方法在bean里没有,编译器不是发觉不了吗？**

**答：** 对，正是如此。而且也正是这一点让大多数Java开发人员有些不放心。毕竟，由接口做出“约定”是Java中接口的一大优点，这样编译器就能保证你已经正确地实现了所有接口方法。

不过，先不要慌！在本书所提供的开发示例中，我们确实必须很仔细，因为编译器不能帮助我们保证已经实现了组件接口的所有业务方法（因为我们没有让bean实现组件接口）。但是在实际应用中往往会使用一个支持EJB的开发环境，而这样一个开发环境自然会确保你提供了所需的方法，这有两种做法：一种做法是，这个开发环境会根据组件接口在你的bean类中放上有具体代码的方法；或者反过来，先找到bean类中的业务方法；再把它放到组件接口中去。至少，大多数服务器都会在你部署bean时或者部署之前进行检查，看组件接口和bean类中的方法是否匹配。

如果你还觉得有问题，别着急，稍后我们会介绍一种技术，利用这种技术就可以绕过这个问题。不过除了很少见的一些情况外，根本不需要使用这种技术。




**bean的命名约定不是EJB规范的一部分。**

Advice, AdviceHome, AdviceBean...

你不必非得给企业bean起这样一些名字！（不过，起这样的名字的的确确是个很好的想法。）要记住，不要被类或接口的名字所蒙蔽，例如，如果有一个名为AdviceHome的组件接口和一个名为AdviceBean的组件接口，这也是完全合法的。在考试中，一定要按照类或者接口的声明来做判断，而不是根据类名草率下结论。

## BRAIN POWER

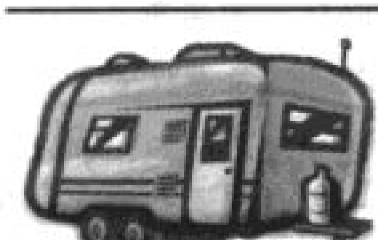
前面你已经看到了，尽管bean必须与组件接口有相同的方法，但是bean并没有实现组件接口。如果顾问bean有以下需求需要解决，请开动脑筋想个办法（办法可能不只一种）：

1. 组件接口（Advice）必须扩展EJBObject。
2. bean必须实现SessionBean接口。但bean不能实现组件接口（Advice）。
3. 我们希望编译器能够验证bean （AdviceBean)有组件接口中同样的方法。

（提示：要正确理解"bean有组件接口中同样的方法"，这与"bean有组件接口中声明的同样的方法"是两码事。）

（提示：这只需要一些Java知识，不要求任何EJB知识。）

---

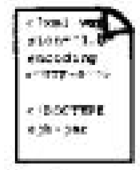


## Off the path

**支持bean的开发工具**

如今，许多EJB程序员都使用了一些支持EJB的开发工具。换句话说，支持bean的IDE很清楚home接口、组件接口和bean类这三大部分之间如何关联。这些工具中有很多还知道如何与一个或多个应用服务器直接对话，所以用这些工具不仅可以完成开发，还可以部署bean，而不是先在一个开发环境中开发，然后再换用服务器自己的部署工具进行部署（服务器提供的部署工具往往不太友好）。EJB开发工具的一大优点是，你不必操心组件接口中的业务方法与具体bean类中的方法是否匹配，反之亦然，即不用担心bean类方法是否与组件接口中的业务方法匹配。一个好的IDE（支持EJB的IDE）会确保二者是一致的。

1 bean
2 接口
**3 XML部署描述文件**
4 ejb-jar
5 部署



ejb-jar.xml

必须用这个名字！部署描述文件只能取名为"ejb-jar.xml"

## 3 创建一个XML部署描述文件，告诉服务器你的bean是什么，以及该如何管理。

（在这本书里，部署描述文件并不是我们自己写的；这个工作是让部署工具帮我们做的。）

部署描述文件（deployment descriptor，DD）描述的是你的bean的结构，包括三个文件（组件接口、home接口和bean类）相互之间如何关联。服务器不会根据你对类或接口的命名来确定哪个是home、哪个是bean。你必须通过部署描述文件告诉服务器到底谁是谁，以及这三者之间是如何连接的。不过，部署描述文件做的还不只如此。有些bean在部署描述文件中的描述竟达几页之多！

对于这个简单的bean，它的部署描述文件则很短。要知道，你无需记住部署描述文件中XML的语法。在本书后面的一些章节中，我们会详细说明你需要知道部署描述文件中的哪些方面。

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems,
Inc.//DTD Enterprise JavaBeans 2.0//EN' 'http://
java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
  <display-name>Ejb1</display-name>
  <enterprise-beans>

    <session>
      <display-name>AdviceBean</display-name>
      <ejb-name>AdviceBean</ejb-name>
      <home>headfirst.AdviceHome</home>
      <remote>headfirst.Advice</remote>
      <ejb-class>headfirst.AdviceBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
      <security-identity>
        <description></description>
        <use-caller-identity></use-caller-identity>
      </security-identity>
    </session>

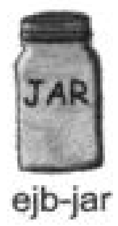
  </enterprise-beans>
</ejb-jar>
```

**如果你使用一个能帮你构建部署描述文件的工具，就不必自己亲自动手来编写XML部署描述文件。**

**可以使用J2EE RI的bean向导工具来做这个工作，它"吐"出来的XML在所有EJB 2.0容器中都能工作！**

现在你只要知道，对于应用中的每个bean，在部署描述文件中都有一个相应的元素，这个元素要描述这个bean的结构和类型，知道这些就足够了。

1 bean
2 接口
3 XML部署描述文件
**4 ejb-jar**
5 部署

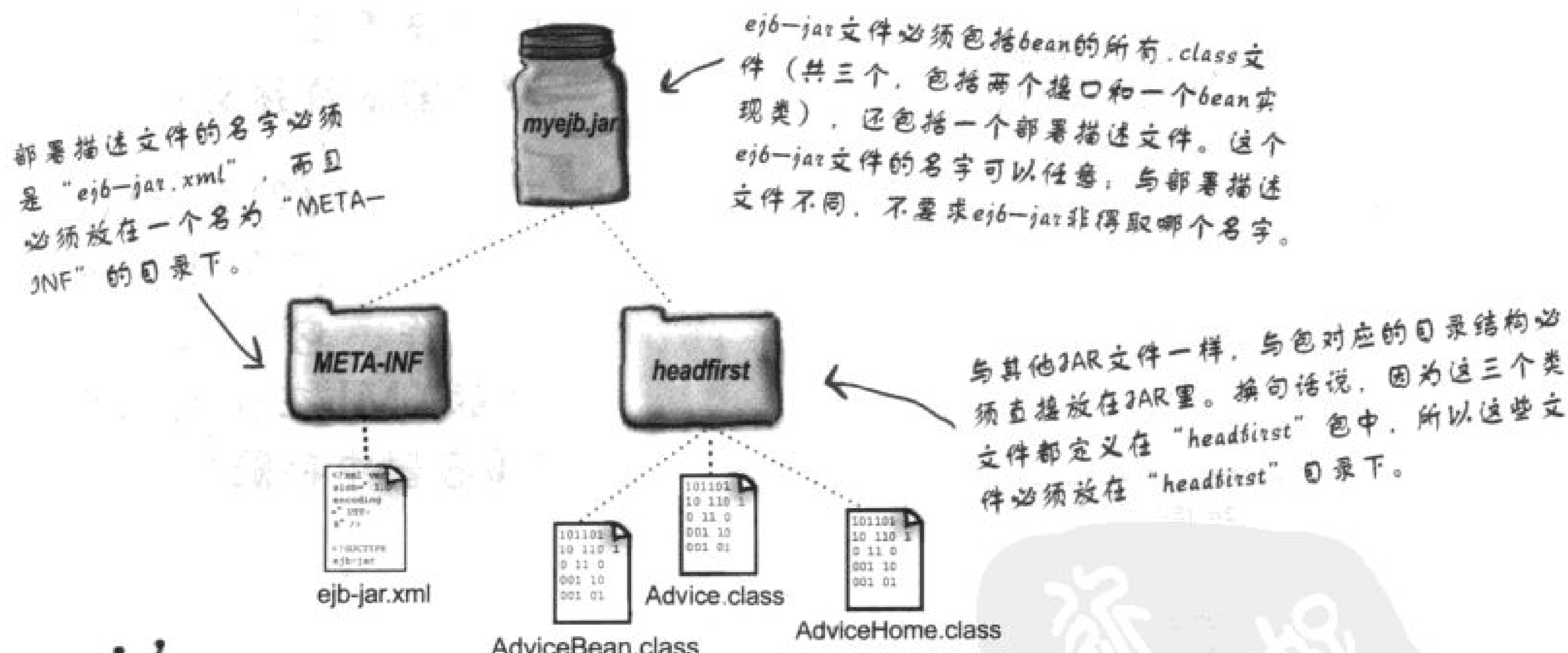



## ❹ 把bean、接口和部署描述文件都放到一个ejb-jar文件里。

（在这本书里，我们不是用JAR工具自己来建ejb-jar；这个工作会让部署工具来做。）

作为一个bean开发人员（正式的说法是bean提供者），肯定要把bean放在一个JAR中。规范指出，ejb-jar就是一个JAR文件，其中包括bean依赖的所有东西（类和接口，以及一个部署描述文件）。

这个工作不用你自己手工来做，因为我们用了RI！我们的做法不是自己编写XML部署描述文件，然后再使用**jar**工具来打包，而是使用了RI的**deploytool**向导工具，这样不仅更简单，而且也不那么容易出错。换句话说，我们要把第4步和第5步合二为一。目前需要知道：在用已编译的类和接口以及部署描述文件建立一个JAR文件之前，bean还不能算是一个bean。

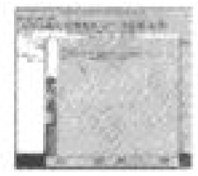





**当心！**

这个考试要求你知道ejb-jar文件里要有什么，不该有什么。容器生成的类和接口（稍后就会介绍）不应该放在ejb-jar文件中！可以把ejb-jar看作是你作为bean开发人员要创建的东西。这就是你交付的成果！容器/服务器也有自己要交付的成果，它们生成的那些东西不能放到ejb-jar里。假设你在一家Beans-R-Us公司工作，而且没有用到支持EJB的服务器。ejb-jar就是要体现你做的工作：bean开发人员的构建块组件（也就是bean）要放在ejb-jar里，这样其他开发人员就可以使用这些bean来组装应用了。你可以用一个服务器工具来帮助创建XML部署描述文件，但是部署描述文件仍要作为你交付的成果（要放在ejb-jar中），而不论到底是谁具体创建的。

1 bean
2 接口
3 XML部署描述文件
4 ejb-jar
5 部署



## ❺ 用服务器开发商提供的工具，把bean部署到服务器上。

你的bean肯定得做点什么，这是迟早的事。这些bean要组装到一个应用中，还要部署到一个服务器上，等着客户来调用。

这一步的工作量很大。实际上我们偷了点懒，因为这其实是两步：应用组装和部署。

### ◆ 应用组装

这一步是指让bean从可重用组件“成长”为应用的一部分。对于简单的bean，这可能就表示只需编写一个能访问这个bean（也就是说，调用这个bean的业务方法）的客户即可。换句话说，一个bean可能就是服务器端应用的全部。不过也可能没有这么简单，在这一步里可能需要把多个bean（和其他Java类）集成到一个定制应用中，这通常意味着要取得多个不同的bean（这些bean有其自己的ejb-jar，而且有自己的部署描述文件），再把这些bean放到一个新的ejb-jar里面，而且这个ejb-jar有惟一的一个部署描述文件，这个部署描述文件会描述两个或多个bean彼此是如何关联的。

在组装这一步，你可能还要向部署描述文件增加一些新的信息，比如那些bean开发人员不知道的东西。例如，bean开发人员可能编写了一些代码，其中要使用一个bean特定的特殊“性质”（也称为一个环境项，这个内容将在本书后面的一章中详细介绍），这个性质表示该应用使用的税费总额。不过，bean开发人员并不知道要给这个税费总额指定什么值才好，所以他可能不会在部署描述文件中为这个性质指定值。然后应用组装人员上场，他会看到（通过阅读部署描述文件）这个bean使用了一个性质，并得出这个性质值应该是多少，然后把这个值加到部署描述文件中去。

对于Advice bean来说，把这个bean放到ejb-jar里，构建部署描述文件，并进行部署，所有这些只需一大步就可以完成。

### ◆ 部署

到这一步才算是把肉放在案板上了，现在bean总算遇到了服务器，开发人员终于和系统管理员打上了照面。部署有两个极其重要的部分，这就是给bean取名字（这样客户才能知道怎么找到它），还要把bean置于容器的控制之下。

规范对于怎么部署bean并没有多说什么；究竟如何部署完全取决于你用的是什么EJB服务器/容器。

# EJB角色和职责




## EJB角色：bean提供者

**交付成果**：ejb-jar文件（包括一个或多个bean和一个XML部署描述文件）

**主要职责**：设计和编写企业javabean。换句话说，就是编写bean代码。

**特点**：知道对于一个特定领域，一个特定类型的组件中应该有什么样的业务逻辑。

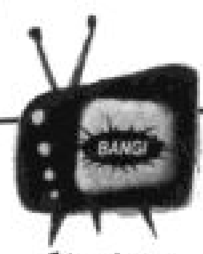



## EJB角色：应用组装人员

**交付成果**：ejb-jar文件（包括一个或多个bean和一个XML部署描述文件，这个部署描述文件中不仅有bean提供者提供的信息，还包括应用组装信息）。还可能创建客户，或者定义与其他组件（如JSP）之间的交互。

**主要职责**：把一个或多个企业bean合并到一个更大规模的应用中。有时应用组装人员可能披着bean提供者的外衣，把新的bean和原有的bean集成在一起构建一个应用。定义应用的安全和事务行为。

**特点**：绝对是领域专家。可能没有bean提供者编写的代码多。



**当心！** **考试会考到不同角色之间的微妙差别。**

对本书中提到EJB角色的地方要特别仔细，尤其是后面更详细地介绍部署描述文件时更要注意。考试时肯定会考哪个角色做什么工作，而且这些题目肯定会有些难度。应用组装人员和bean提供者在许多方面都存在重叠，另外部署人员和应用组装人员也不是泾渭分明的。对考试来说，你必须知道某个任务是哪个角色的主要职责（通常与部署描述文件信息有关）。这些内容很是麻烦，这一页和下一页的介绍还只是个开头……

## EJB角色：部署人员

**交付成果**：针对一个特定操作环境进行定制的企业bean，而且已经部署到服务器上。

**主要职责**：拿到应用组装人员交付的成果，研究ejb-jar中的部署描述文件，解析得出所有外部依赖关系。例如，如果bean依赖于一个特定资源，部署人员就必须把bean提供者指定的逻辑名映射为服务器上资源的实际名。要记住，Bill写bean代码时并不知道这个代码最后要部署到Dick的服务器上。Bill必须为数据库建立一个“虚构的”名字，Dick则必须把这个自创的名字与一个真实的资源绑定起来。

**特点**：他是特定操作领域的专家。他通晓这个系统的安全用户和角色，知道服务器上已经部署了什么东西，而且了解怎么解释bean提供者和应用组装人员提供的部署描述文件信息。







## EJB角色：容器和服务器提供者

**交付成果**：EJB 2.0兼容服务器、部署工具、企业bean的运行时环境。

**主要职责**：实现规范。

**特点**：他们是分布式对象和事务以及其他底层系统服务领域的专家。

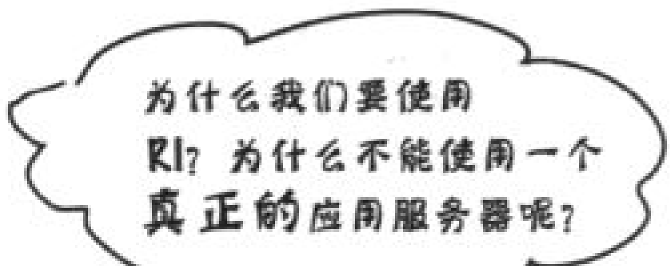
# 教程：

## 下面来建立、部署和测试AdviceBean。

我们会编写代码、进行编译、启动服务器、启动deploytool，再使用deploytool建立部署描述文件和ejb-jar，部署bean，创建一个客户，然后使用这个客户来测试前面部署的bean。我们惟一没有做的事情只是安装和配置服务器。这里假设你已经完成了服务器的安装和配置。

如果你还没有把J2EE 1.3 RI启动和运行起来，请访问http://java.sun.com/j2ee/，下载J2EE的1.3版本（其中包括安装说明），然后再回到http://java.sun.com/j2ee/，下载J2EE API文档。

要记住，这个考试是面向J2EE 1.3而不是1.4的！不管你想怎么做，有一点绝对要避免，这就是千万不要使用1.4规范来为考试做准备。那么为什么考试使用的是1.3版本而不是1.4呢？这个原因在引子中已经做过详细解释，简单地说可以总结为：如果一个人掌握的技术几乎没有人使用，这可不是我们想要认证的东西。认证不是说“我了解这个技术最新、最好的版本”，认证的意义在于“我了解人们现在使用的这项技术。这个技术我已经使用至少6个月了”。






我要使用哪个服务器呢？在学习和实践中我们采用的是RI，因为对于你要使用哪一个服务器我们一无所知，而且RI是所有能免费得到的服务器中最简单的一个。我们希望服务器尽可能地容易，这样你就能把精力集中在EJB技术上，而不是过多地考虑特定工具完成的任务。

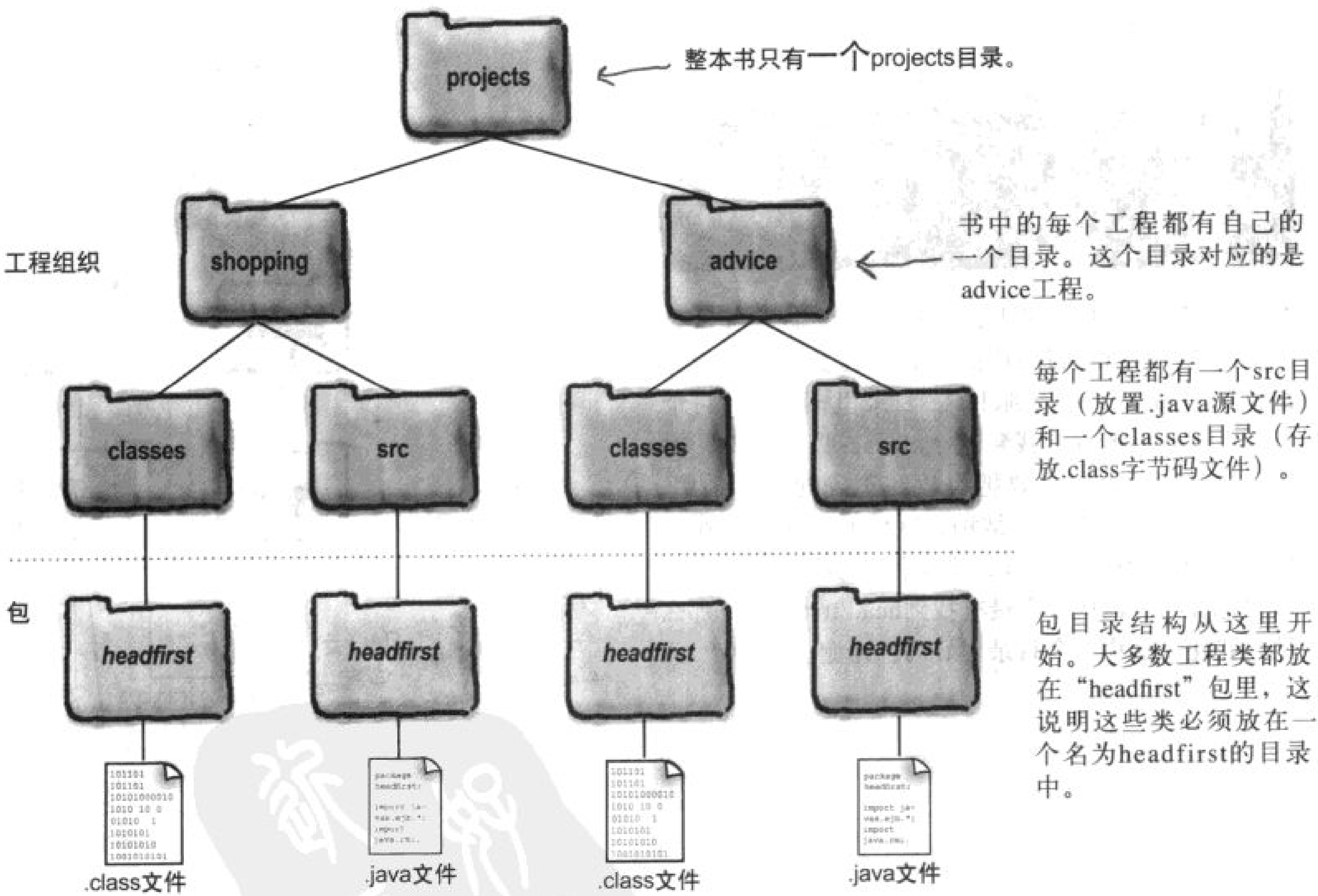
有一些开源产品（如JBoss）同时也是真正的成品服务器，所以如果采用这样一个服务器，需要处理的配置和管理任务就会多得多。使用RI的话，你就能把更多的时间用在你必须做的工作上，而不论服务器是什么，同时，只需花最少的时间来学习服务器是用什么方法来完成这些工作的。

**Relax** 考试中不会涉及有关J2EE RI的内容。

也不会考有关其他应用服务器的内容。你不需要了解开发商提供的任何用于开发或部署的工具（包括Sun提供的此类工具）。你要知道的只是每一个EJB兼容服务器都应该有的功能，而开发商特有的特性或配置细节你无需了解。

# 组织工程目录

本书里的所有bean都组织到包中，这说明在编译和运行时必须更加仔细一些。这一章中我们给出了许多指令，**每个指令都假设你的工程是按以下所示的方式来组织的**。如果你没有采用这个结构，就必须根据你自己的目录结构对我们提供的命令和部署步骤自行调整。




## 安排终端/命令行指令

File Edit Window Help Accessorize

```
%cd ~/projects/advice/src
```

从src目录**编译**

```
%cd ~/projects/advice
```

从特定的工程目录**运行**客户

# 编译两个接口和bean类

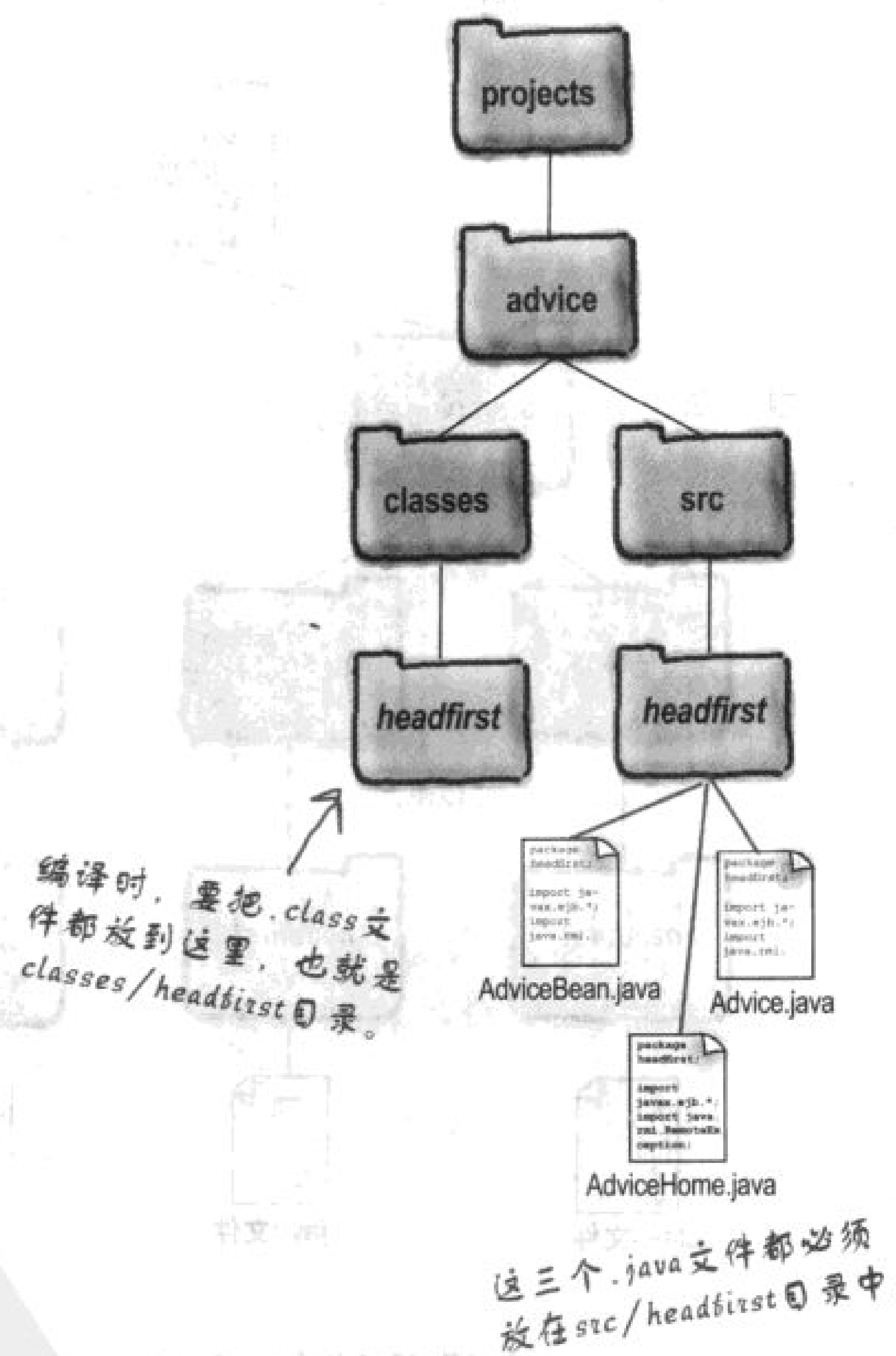
到目前为止，我们已经编写了两个接口和一个bean类，但是除此以外还需要对它们进行编译。在此之后，我们会建立ejb-jar（其中包括的是类文件，而不是源文件）。

```
File Edit Window Help WhyAmIHere
%cd ~/projects/advice/src

%javac -d ../classes headfirst/*.java
```

我们使用了-d编译标志，这样上面的命令行就表示"编译headfirst目录中的所有.java文件，然后把编译后的.class文件放到classes目录里，从当前目录（src）向上一级就能找到这个classes目录。哦，对了，差点忘了，一定要把类放在正确的包目录中。在classes目录中查找包结构，这说明你应该能找到一个名为headfirst的目录，类文件就要放在这里……如果你在classes目录中没有找到headfirst目录，那么请为我建立这样一个目录，好吗？谢谢。"

此时此刻，你的projects目录结构应该如下所示：



## 启动服务器

为服务器打开一个新终端。让服务器一直运行着，我们希望服务器运行过程中能看到输出，所以不要用这个终端做其他的事情。设置advice目录为你的工作目录。

```
File Edit Window Help BlahBlahBlah
%cd projects/advice
%j2ee -verbose
```

## 你会看到下面的输出

-verbose标志会在终端上打印一堆东西出来（这个标志并不是必要的，不过我们很喜欢用它）。

```
File Edit Window Help WhatThe...?
%j2ee -verbose
J2EE server listen port: 1050
Naming service started:1050
Binding DataSource, name = jdbc/DB1, url = jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = jdbc/InventoryDB, url = jdbc:cloudscape:rmi:
CloudscapeDB;create=true
Binding DataSource, name = jdbc/DB2, url = jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = jdbc/EstoreDB, url = jdbc:cloudscape:rmi:
CloudscapeDB;create=true
Binding DataSource, name = jdbc/Cloudscape, url = jdbc:cloudscape:rmi:
CloudscapeDB;create=true
Binding DataSource, name = jdbc/XACloudscape, url = jdbc/XACloudscape__xa
Binding DataSource, name = jdbc/XACloudscape__xa, dataSource = COM.cloudscape.core.RemoteX
aDataSource@4c6715
Starting JMS service...Initialization complete - waiting for client requestsBinding: < JMS
Destination : jms/Topic , javax.jms.Topic >Binding: < JMS Destination : jms/Queue , javax.
jms.Queue >Binding: < JMS Cnx Factory : QueueConnectionFactory , Queue , No properties
>Binding: < JMS Cnx Factory : jms/TopicConnectionFactory , Topic , No properties >Binding:
< JMS Cnx Factory : jms/QueueConnectionFactory , Queue , No properties >Binding: < JMS Cnx
Factory : TopicConnectionFactory , Topic , No properties >
Starting web service at port: 8000
Starting secure web service at port: 7000
J2EE SDK/1.3.1
Starting web service at port: 9191
J2EE SDK/1.3.1
J2EE server startup complete.
```

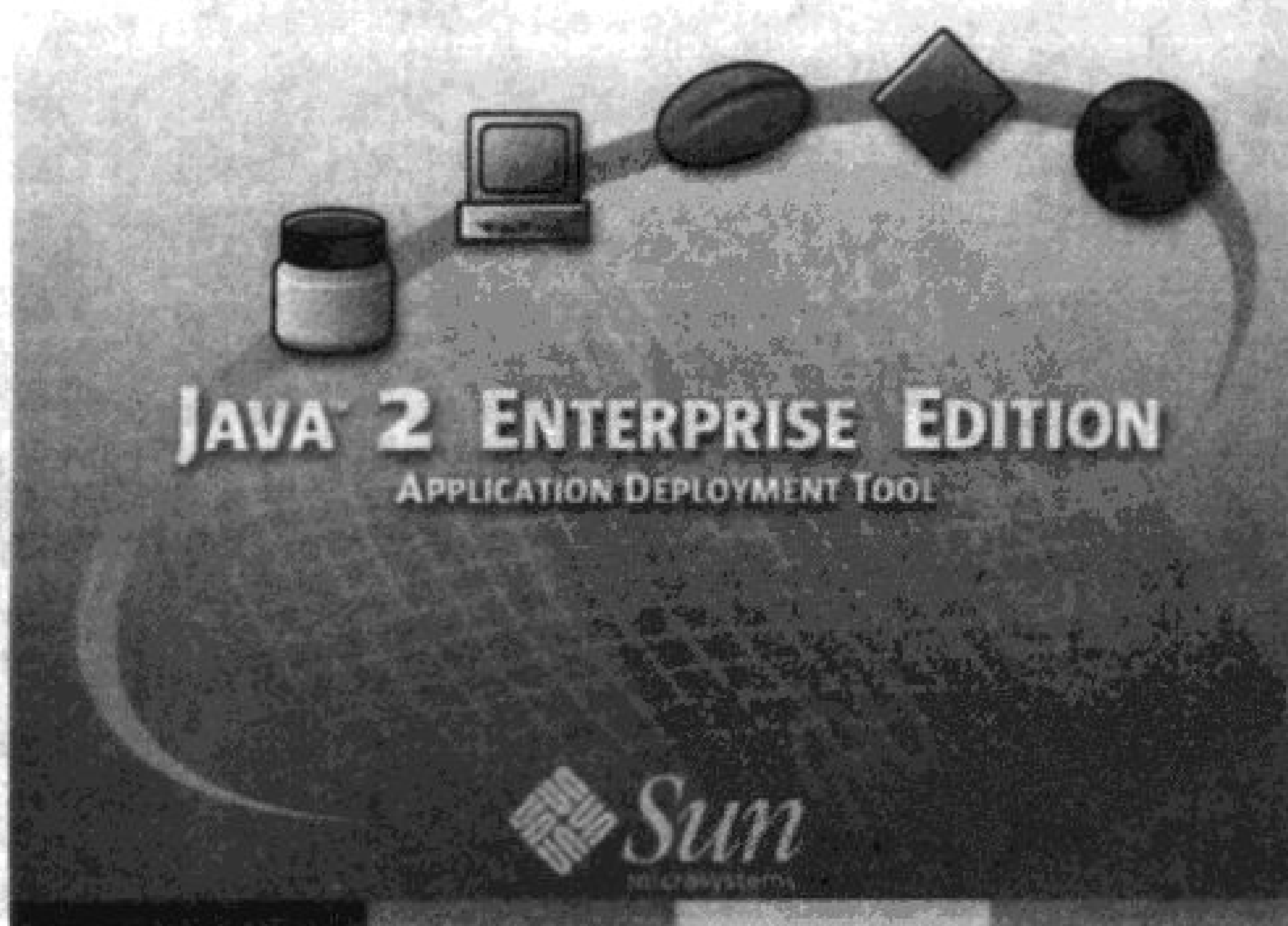
# 启动deploytool

为deploytool打开一个新终端。这个工具是J2EE RI的一部分，它提供了创建ejb-jar和部署描述文件所需的所有功能，并能完成最后的部署，即把bean部署到RI服务器上。

```
File Edit Window Help Chill
%deploytool
Starting Deployment tool, version 1.3.1
(Type 'deploytool -help' for command line
options.)
```

# 你会看到下面的输出

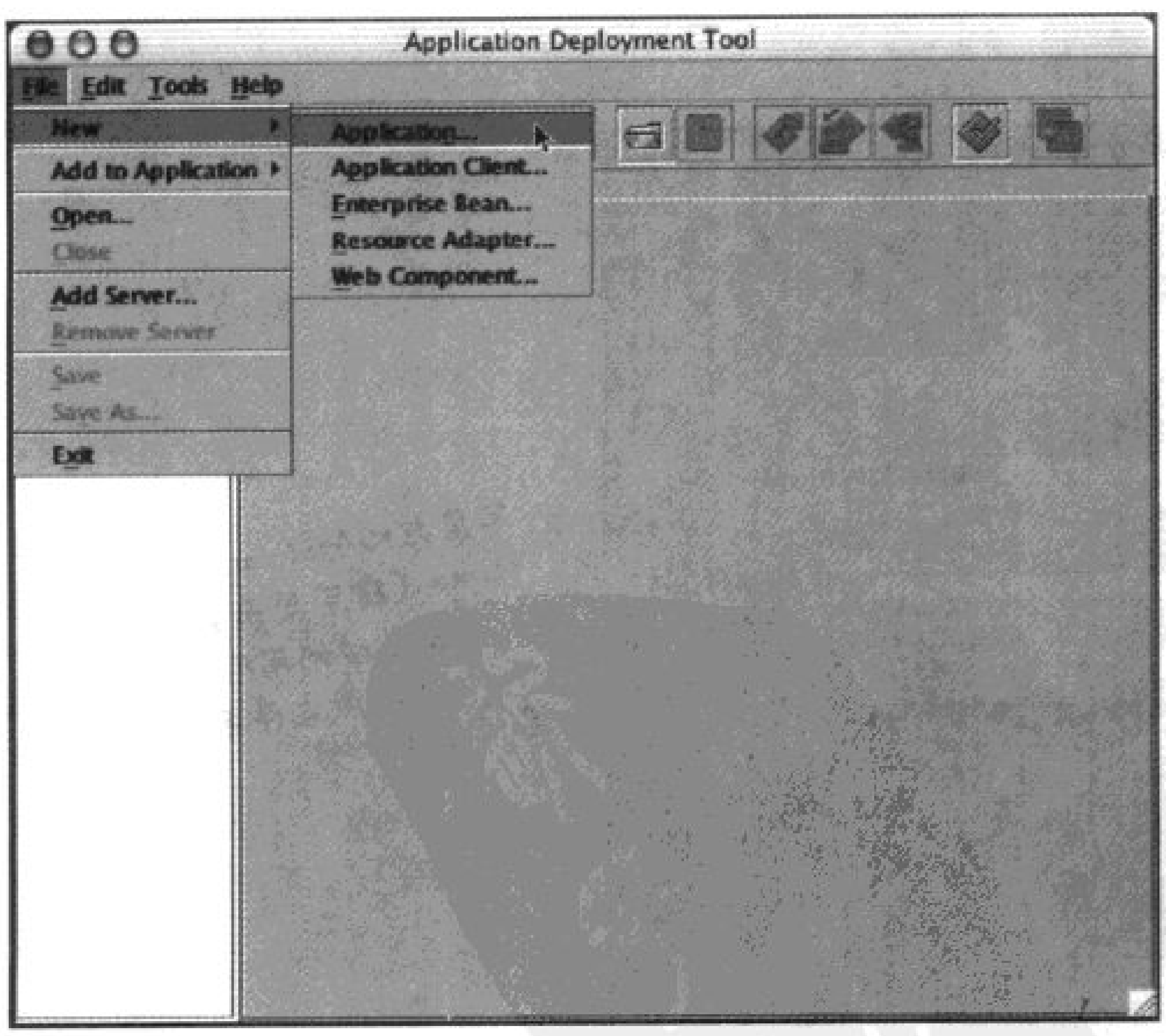
会弹出一个可爱的启动界面，应用加载时它会一直留在屏幕上。如果点击这个启动界面，它就会消失，所以如果看上去什么都没有发生，你也不要惊慌。耐心一点。



## 建立一个新应用

要记住，RI是一个J2EE服务器，它不只是一个EJB容器。所以在建立ejb-jar和部署应用之前，还需要做一些有关J2EE的工作。我们要在这一步创建一个新的J2EE应用，对现在来说，可以这样来看J2EE应用：它把bean包起来，并为服务器稍微增加了一些信息。J2EE应用和EJB应用之间的主要差别在于，J2EE应用除了可以包括EJB组件外还可以包括Web组件（servlet和JSP），所有这些都可以作为一部分集成在一个应用中。

**选择 File ▶ New ▶ Application**

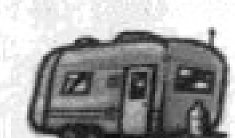



there are no Dumb Questions

**问：这是不是意味着，如果我想同时使用servlet和EJB，就必须有一个J2EE服务器？**

**答：** 并非如此。利用J2EE服务器的话，Web组件和EJB组件会集成得更紧密，这说明，所有组件都能考虑到彼此的事务和安全性。不过，你也完全可以用一个servlet作为企业bean的客户，即使这个bean不在同一个应用服务器上运行（甚至不在同一个物理主机上）也不碍事。J2EE服务器还有一个好处，这就是可以很容易地把两种组件（Web组件和EJB组件）都部署为一个企业应用的一部分。

说了这么多，还需要知道，如果你在构建EJB 2.0应用，极有可能是在J2EE服务器上运行这些应用。要记住，如今很少有独立的EJB容器。几乎所有主要开发商都在一个J2EE服务器中运行自己的EJB容器。



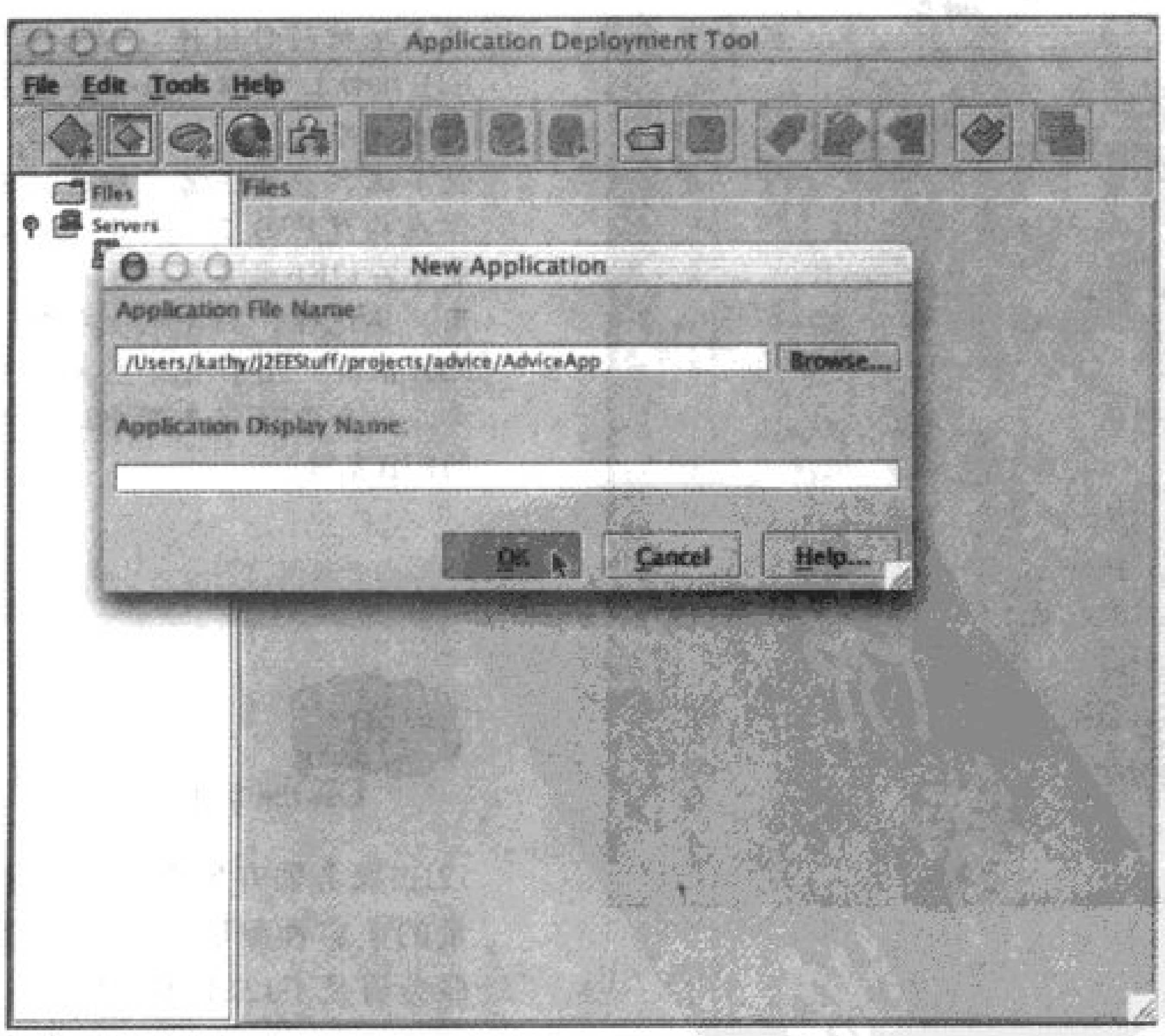
Off the path

J2EE服务器开发商必须通过大量的兼容性测试，才能称这个服务器是“J2EE兼容的”。例如，J2EE 1.3服务器必须包括一个EJB 2.0容器，而且这个容器必须实现EJB 2.0规范。

# 给新应用取名并保存

这部分有点麻烦。可以使用Browse（浏览）按钮在你自己的目录树中导航，不过，要给应用取名并保存起来，最容易的办法是键入你要创建的文件的完整路径。我们要建立的不是bean本身，可以把它看作为更像是一个文档，其中包含了有关应用的所有信息。在这里有一个约定，我们会在对应于这个特定工程的projects/[某]目录下保存应用。对于Advice bean，这意味着要把应用保存在projects/advice目录中。如果从projects/advice目录启动服务器（换句话说，如果advice正是你的当前工作目录），就能默认地得到正确的名字和位置。

把应用取名为**AdviceApp**
如果需要，要包括projects/advice/AdviceApp 的完整路径
点击 **OK**

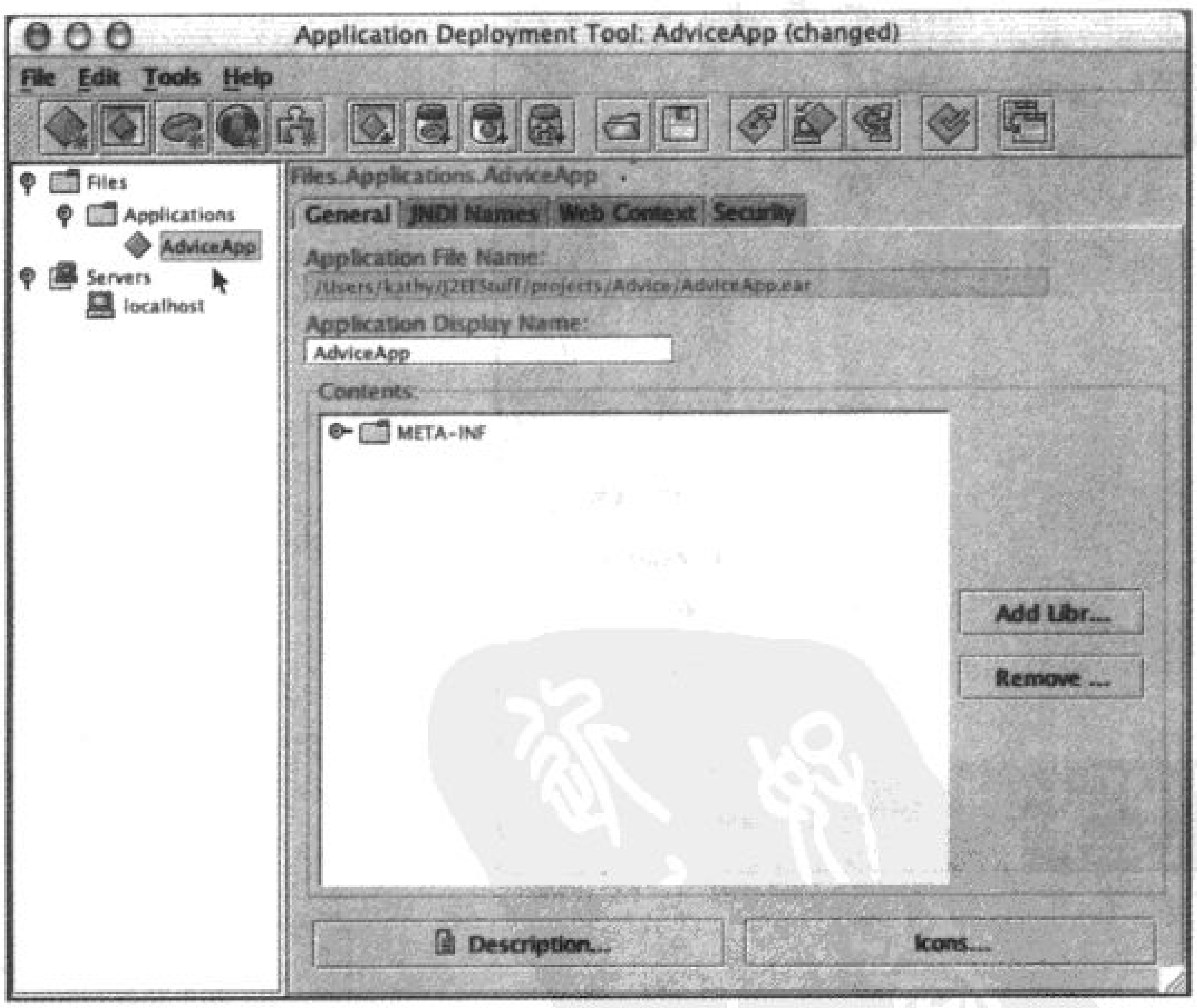



不要担心Application Display Name（应用显示名）这个域；工具会自动地用Application File Name（应用文件名）域的文件名填写这个域。

# 创建应用并对应用命名之后看到的输出

现在，再回到deploytool的主界面。可能必须点击Files或Applications图标将其扩展，不过你会看到，工具已经创建了一个Applications目录，其中有一个名为AdviceApp的目录。点击AdviceApp图标，你会看到有关这个应用的信息，包括应用名、位置和当前内容。此时，除了一个META-INF目录外，这个应用中没有别的东西，META-INF目录保存了有关应用的更多信息，不过我们一般不需要了解这些额外的信息。

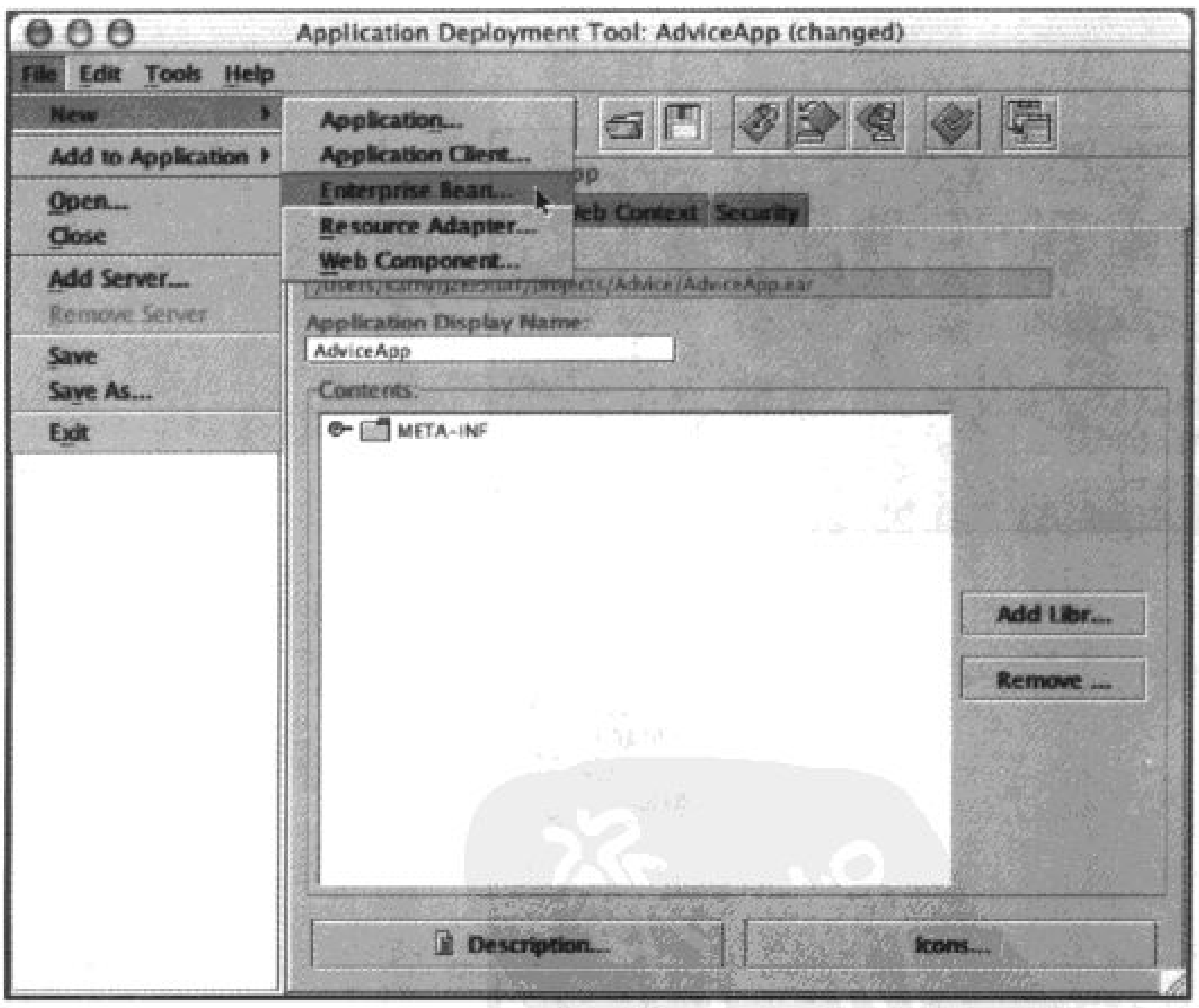
**点击 AdviceApp 图标**



# 下面来建立新的企业bean (ejb-jar和部署描述文件)

这是我们真正要做的——建立具体的bean。前面的各个步骤（建立J2EE应用）只是为了满足J2EE RI的要求，因为我们必须在一个J2EE应用中部署bean。

**选择 File ▶ New ▶ Enterprise Bean...**

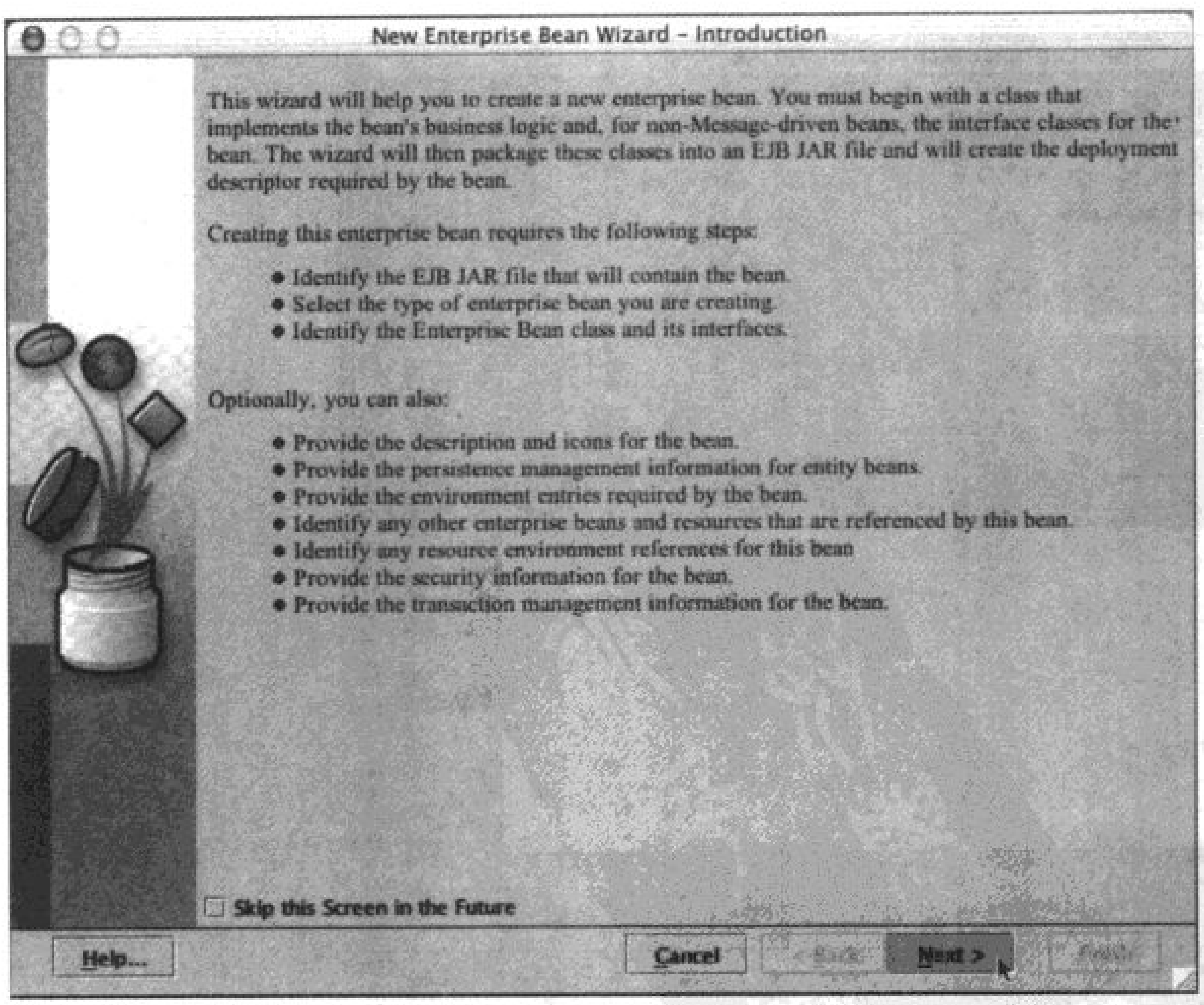


# 下面看到的是一个非常棒的NewEnterprise Bean（新建企业bean）向导工具

所有工作几乎都在deploytool的这一步完成！我们要做的几件关键的事情是：

- 创建ejb-jar
- 把bean类和两个接口放到ejb-jar中
- 创建描述这个bean的部署描述文件

点击 **Next >**

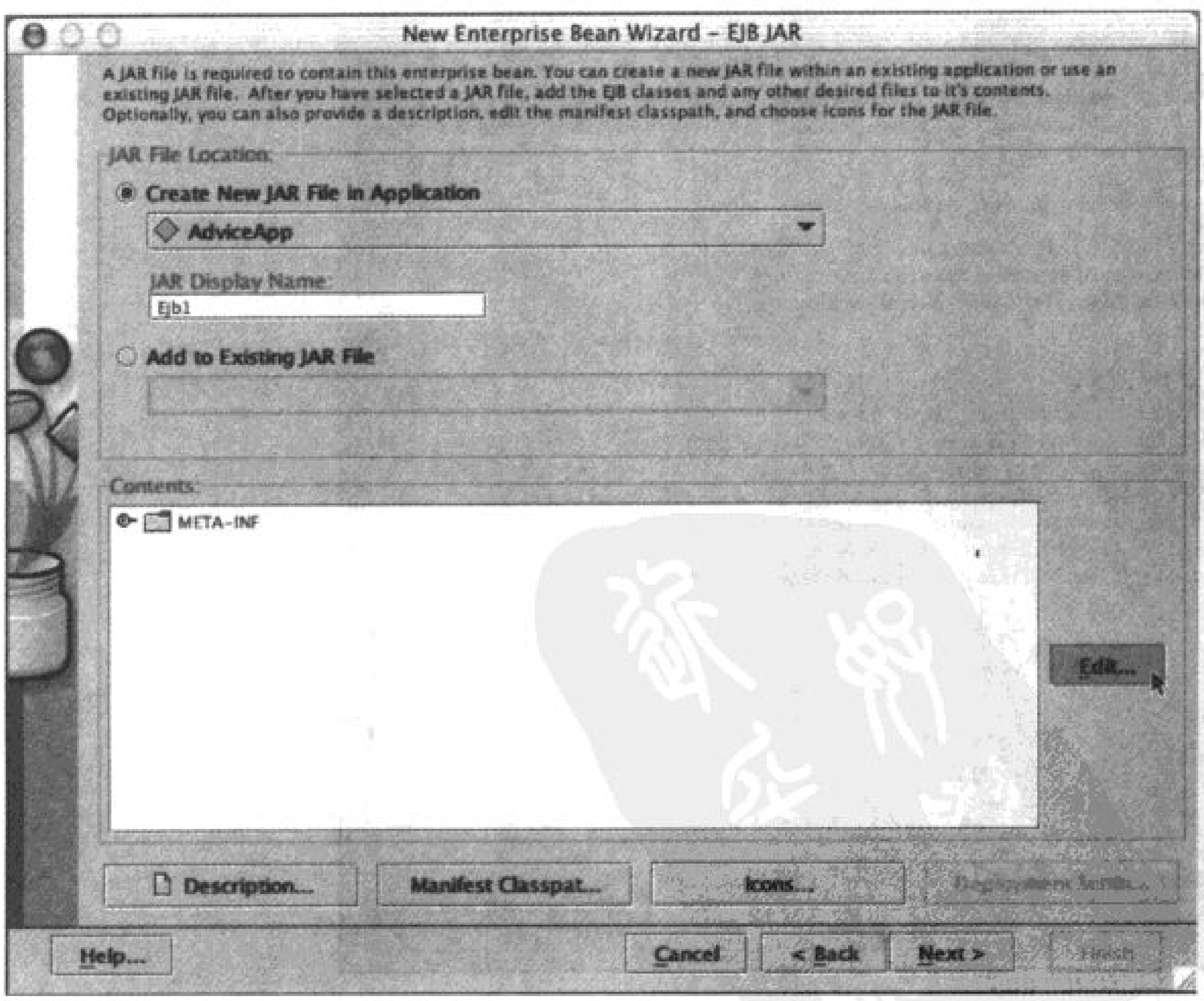


# 创建一个新的ejb-jar

现在，只需接受向导给出的默认选项就可以了。界面左上部的单选按钮显示出，你正在AdviceApp应用中建立一个新JAR。需要注意，AdviceApp是一个下拉列表的一部分，如果应用中还有其他的 ejb-jar，那么还可以选择把这个新bean放在某个已有的JAR中。

部署工具会选择一个很有帮助的显示名“Ejb1”，完成之后，你会在deploytool主界面上看到这个显示名。在实际的应用中并不使用这个名字，所以这个名字到底是什么意义不大，不过，如果一个应用中有多个JAR，你可能希望起一个更有描述性的名字（比如Cart JAR、Account JAR等等）。

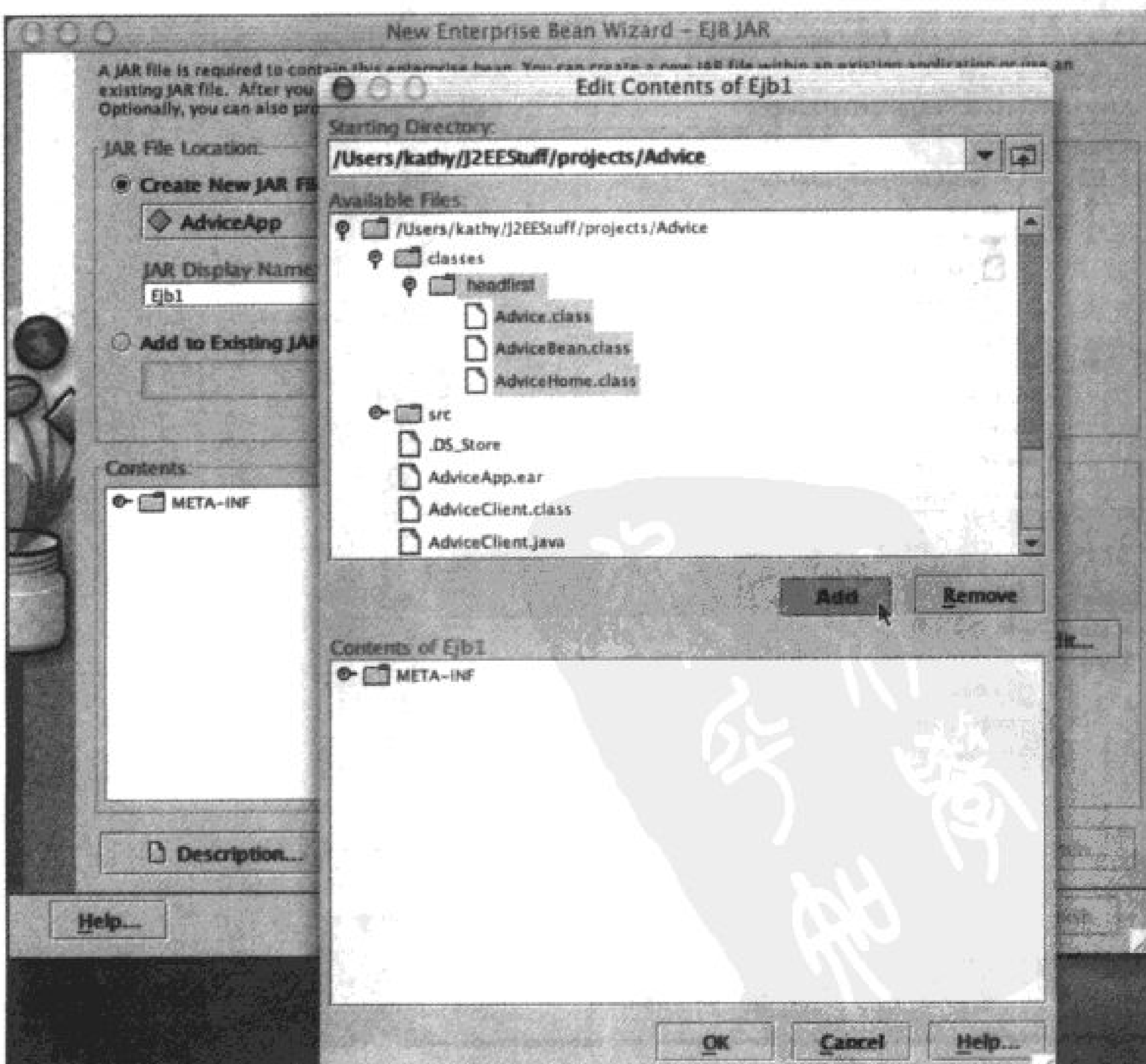
**点击 Edit...**



# 把三个类文件（包括其包目录）增加到JAR中

这是整个过程中最重要的部分！换句话说，千万别搞错了。关键是要把正确的类放到JAR中的包目录结构中，而且只能放在其包目录结构中。也就是说，如果把这三个类文件放到了没有headfirst目录的JAR中，你的bean将无法部署。或者，如果除了headfirst目录还包括了classes目录，你的bean也无法部署。要记住，ejb-jar还是一个JAR文件，所以一般的有关包结构的JAR规则在此也同样适用。

**导航到 Advice 目录**
**扩展 classes 目录，找到 headfirst 目录**
**选择 headfirst 目录**
**点击 Add**

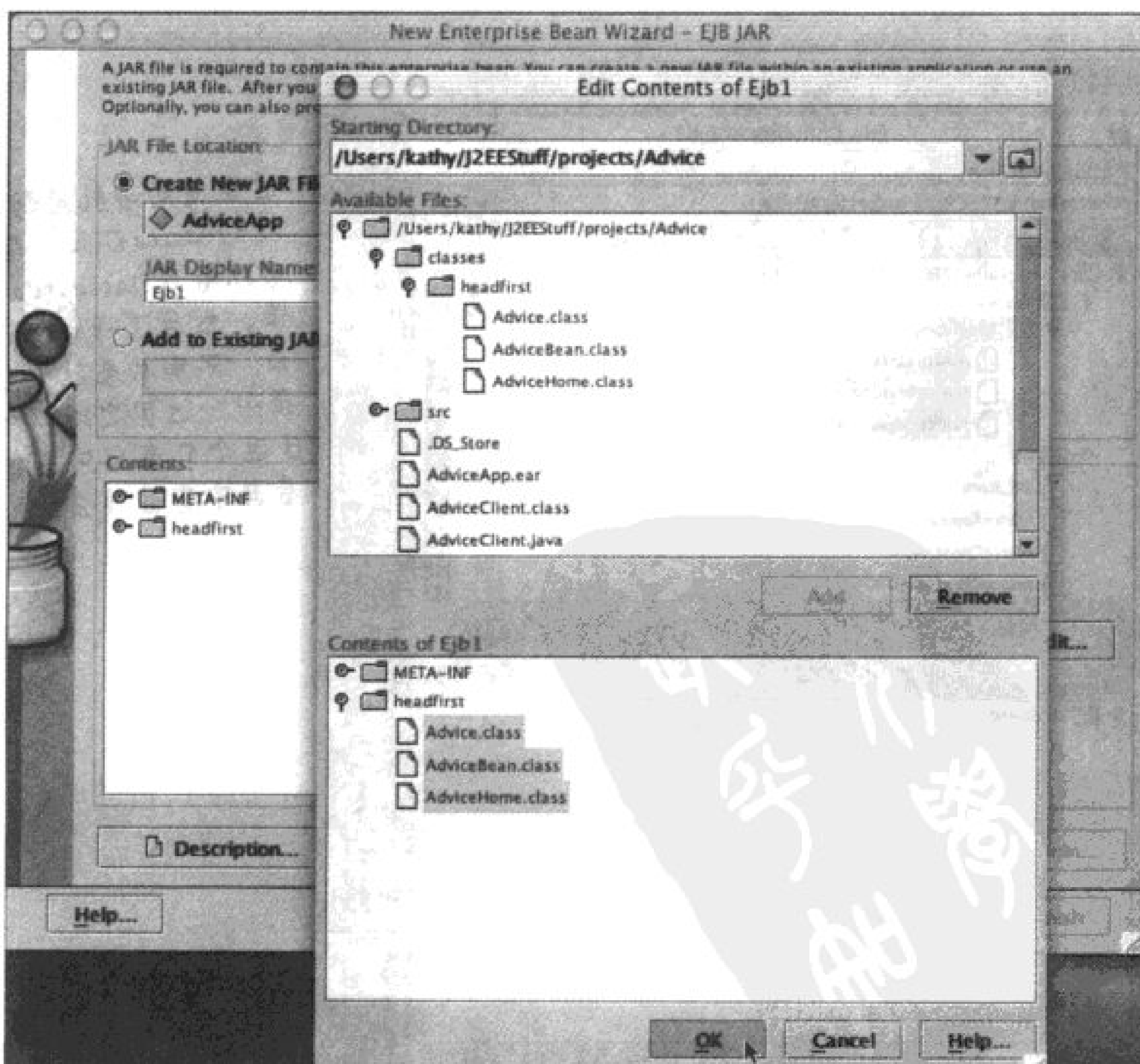



这里使用导航控制图标找到Advice目录，在这里可以看到classes和src目录。然后选择headfirst目录。不用扩展这个目录……这里之所以扩展了这个目录，只是想看看其中有些什么。

# 确认只增加了包目录和类文件

这一步要做对。请看下面标题为"Contents of Ejb1"的窗口，确保JAR中除了META-INF目录外就只有headfirst目录（包括目录的内容）。我们常常见到这样一个经典错误（不只是你会犯这个错误），即只增加类文件，而没有增加包目录。要确保有headfirst目录（而不是classes目录），其中包括三个类文件。还有一个常见的错误是增加了源文件而不是类文件。所以即使你犯了这个错误，也不要太过自责，这个错误大家都很容易犯。

**确保类（和包目录）是正确的**
**如果并非如此，从下面的窗口选择类和包目录，点击 Remove（删除），再重新选择**
**完成后点击OK，再点击 NEXT**



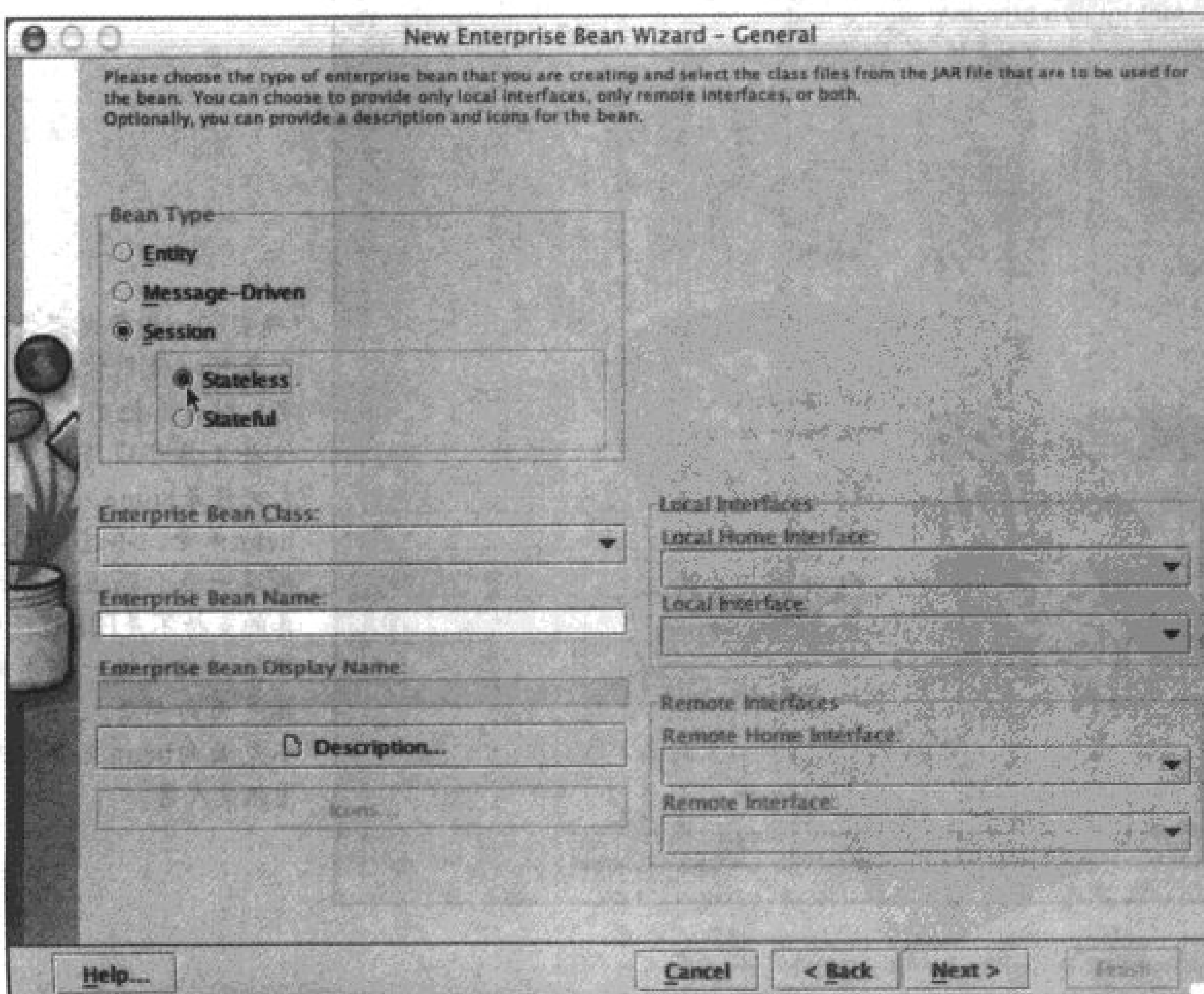
# 让它成为一个无状态会话bean

下面，要向工具提供bean的“结构”信息了。这是一种什么类型的bean，哪个类文件是它的home接口，等等。要记住，工具会使用这些信息来创建部署描述文件。而且EJB容器要使用部署描述文件来得出如何部署和管理这个bean。

Advice bean很简单，客户调用远程对象的一个方法，远程对象返回一个值，然后远程对象会忘记刚才发生的所有事情。故事到此为止。这种情况非常适合采用无状态会话bean解决方案。如果Advice bean需要记住它向客户给出的建议，这个客户以后再调用这个bean时，它要使用以前记住的这一信息，那就应该把这个bean建为一个有状态会话bean。不过既然不需要记住以前给的建议，所以我们不会让它作为有状态会话bean。

随后你会了解到，让Advice bean作为一个实体bean或消息驱动bean是没有意义的。不过，现在让它作为其他类型的bean（而不是会话bean）也为时过晚了，要知道，你的bean类实现的是SessionBean接口，所以实际上已经确定了bean的类型。不过，这个会话bean到底是有状态的还是无状态的，这一点确定起来可能有些难度。现在我们的做法是把它置为无状态的。

**选择Session单选按钮**
**选择Stateless单选按钮**



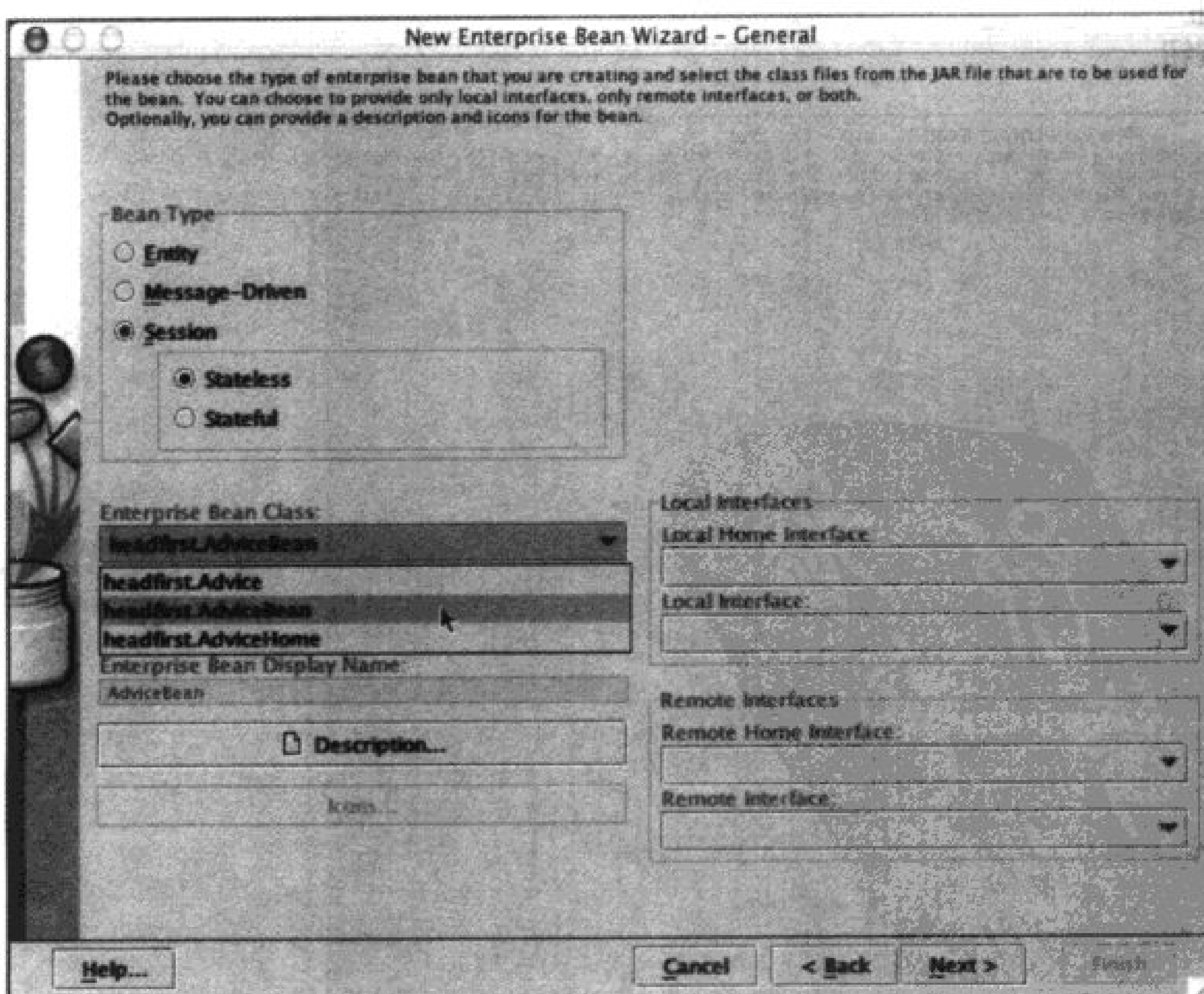

不要着急点击Next按钮！在这个界面上还有工作要做。

# 指出JAR中的三个类文件哪一个是具体的BEAN类

JAR中有3个类文件：home接口（AdviceHome）、组件接口（Advice）和bean（AdviceBean）。不过，EJB容器还需要知道到底谁是谁。要记住，除了你，命名约定对任何人都不能代表什么！容器不会分析JAR中的三个类，不会识别出AdviceHome肯定是home接口，Advice肯定是组件接口，肯定AdviceBean是bean等等*。不是这样的，命名约定只对你和使用你的bean组件的人有意义，对容器是没有意义的。

所以，现在必须告诉工具哪一个文件是bean类（下一页将说明怎么指出两个接口）。然后，工具会把这个信息放到部署描述文件中去。

**点击 Enterprise Bean Class 下拉菜单**
**选择 headfirst.AdviceBean**




*确实有一些开发商提供的工具可以使用你的命名约定，结合其他的一些信息，这些工具可以得出哪一个类文件是home，哪一个是bean等等。不过，RI不支持这一点，而且对开发工具的支持不是EJB规范的一部分。规范惟一要求的工具是面向部署人员的，而不是面向bean提供者或应用组装人员。

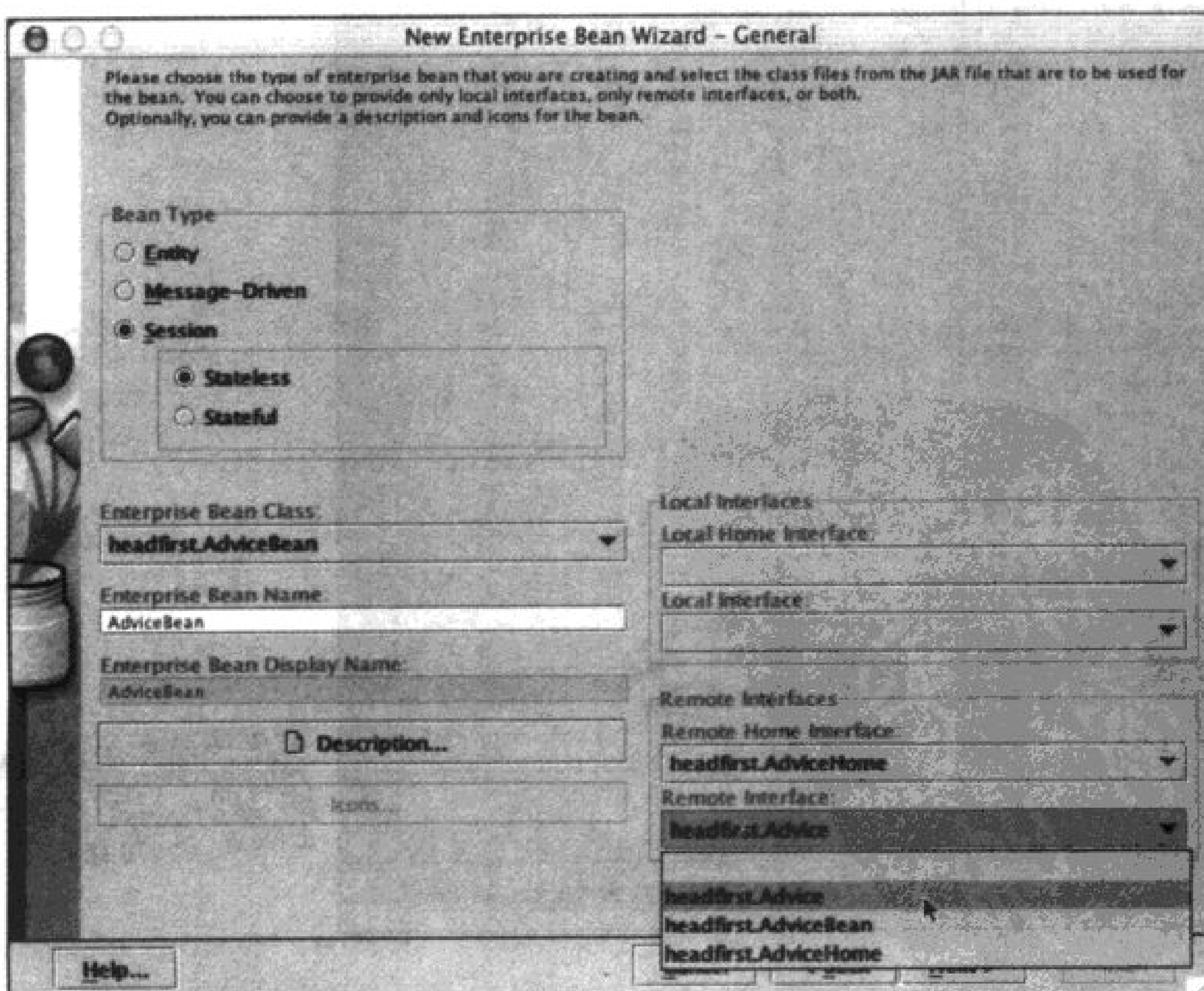
# 指出哪个是home接口，哪个是组件接口

下面，要把对bean类做的事情再做两次，不过，现在要告诉工具哪一个是home接口，哪一个是组件接口。你可能注意到了，这里有两个不同的位置来选择接口，本地接口（Local Interface）和远程接口（Remote Interface）。由于这个bean是远程的（这说明它是一个Java RMI 远程服务，不过这个内容在下一章再深入讨论），所以我们只会用到底下有关远程接口的部分。这里不用去管本地接口部分。

这个界面有一个奇怪的地方，它使用了远程接口而不是远程组件接口（Remote Component Interface）。这实际上是deploytool的一个不好的界面设计（这也是EJB以前版本的产物，因为EJB以前的版本中只有远程接口。本地接口是EJB 2.0新增的）。不过我们应该清楚这一点，不能落后于EJB的发展。

点击**Remote Home Interface** 下拉菜单
选择**headfirst.AdviceHome**

点击**Remote Interface**下拉菜单
选择**headfirst.Advice**



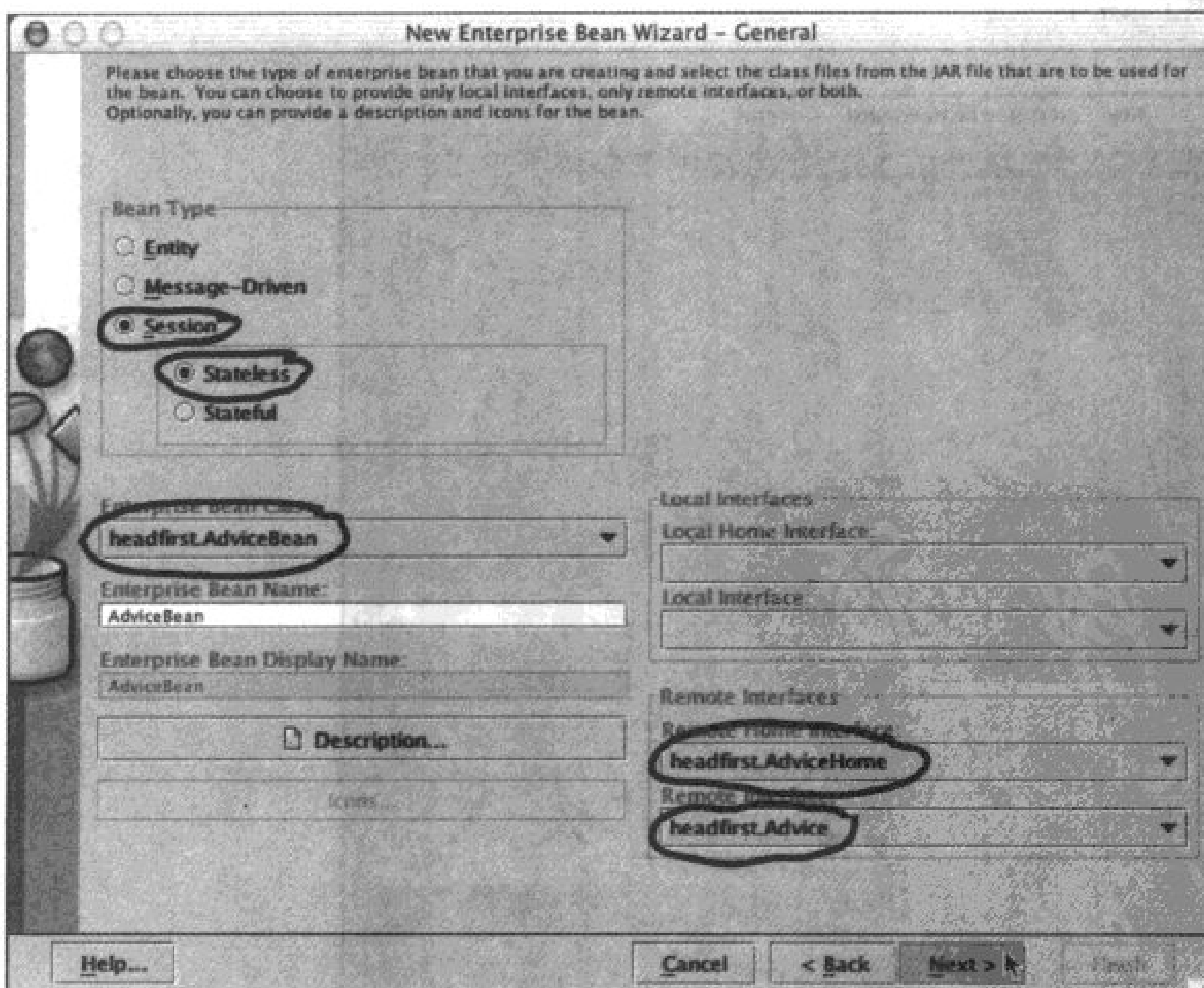
# 验证这个界面上的所有信息！

选择bean类、home接口和组件接口是一个至关重要的决定！一旦完成了余下的bean向导工作，而且创建了部署描述文件，你就必须按这个部署描述文件行事了。如果你不小心把home接口和组件接口弄反了，你的bean将无法工作。在RI中，甚至不能部署（有些服务器允许部署一些结构不正确的bean，这说明这些bean到运行时才会暴露出问题来，而RI则提前就把这扇门关上了，如果bean结构信息不正确，RI甚至不允许它在服务器上部署）。

**确保有如下设置：**

企业**bean**类：**headfirst.AdviceBean**
远程home接口：**headfirst.AdviceHome**
远程接口：**headfirst.Advice**

点击 **NEXT**



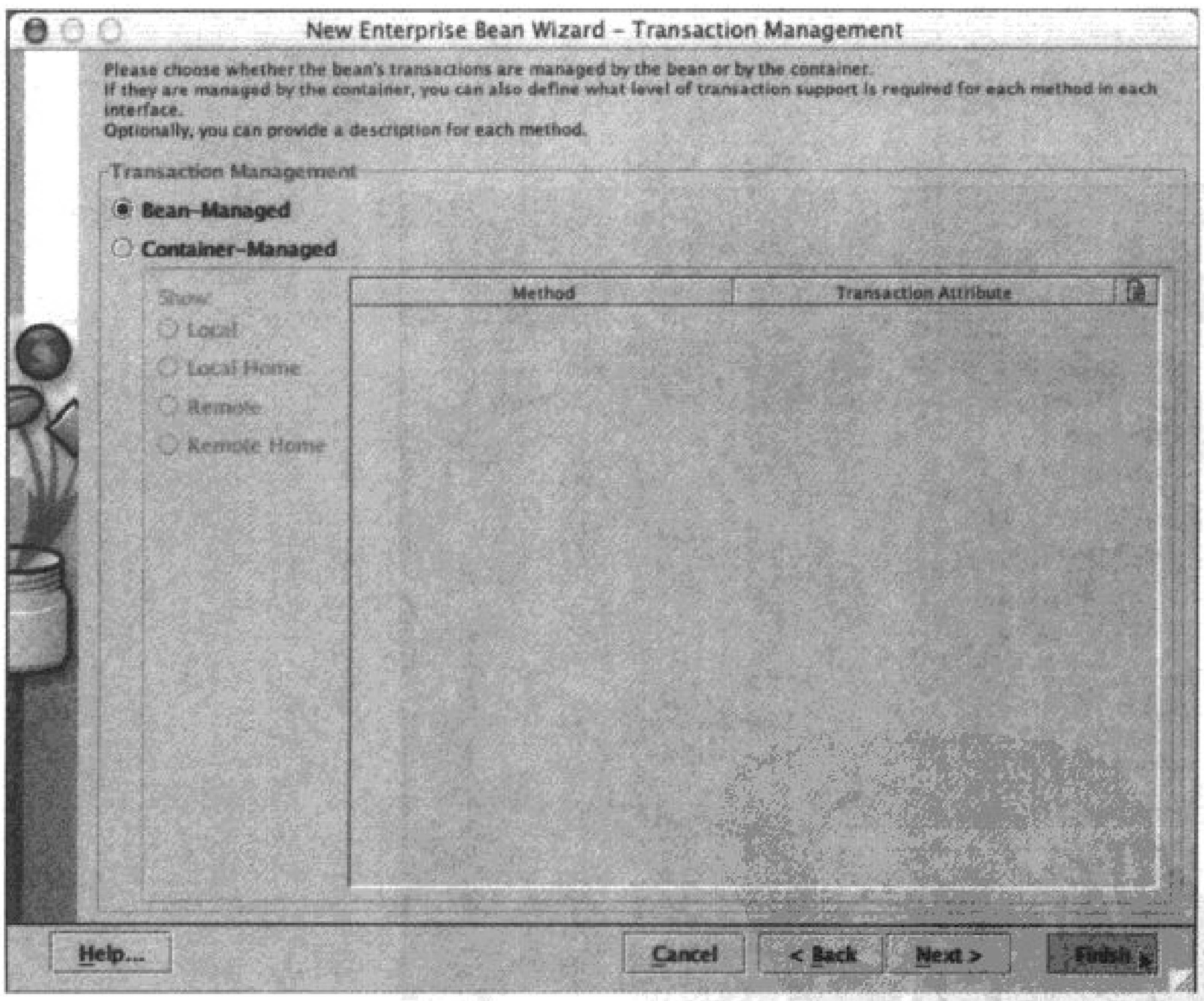

确保你的界面应该是这样的，之后才能点击Next > 按钮！

# 搞定，点击Finish

因为这个bean实在是太简单的，所以我们不关心事务、安全、环境项和数据库访问，对于建立部署描述文件，并把它连同类文件一起放在一个JAR中，我们已经做了该做的所有事情。

所以，你可以忽略Transactions Management（事务管理）界面，不过稍后就要熟悉它了。

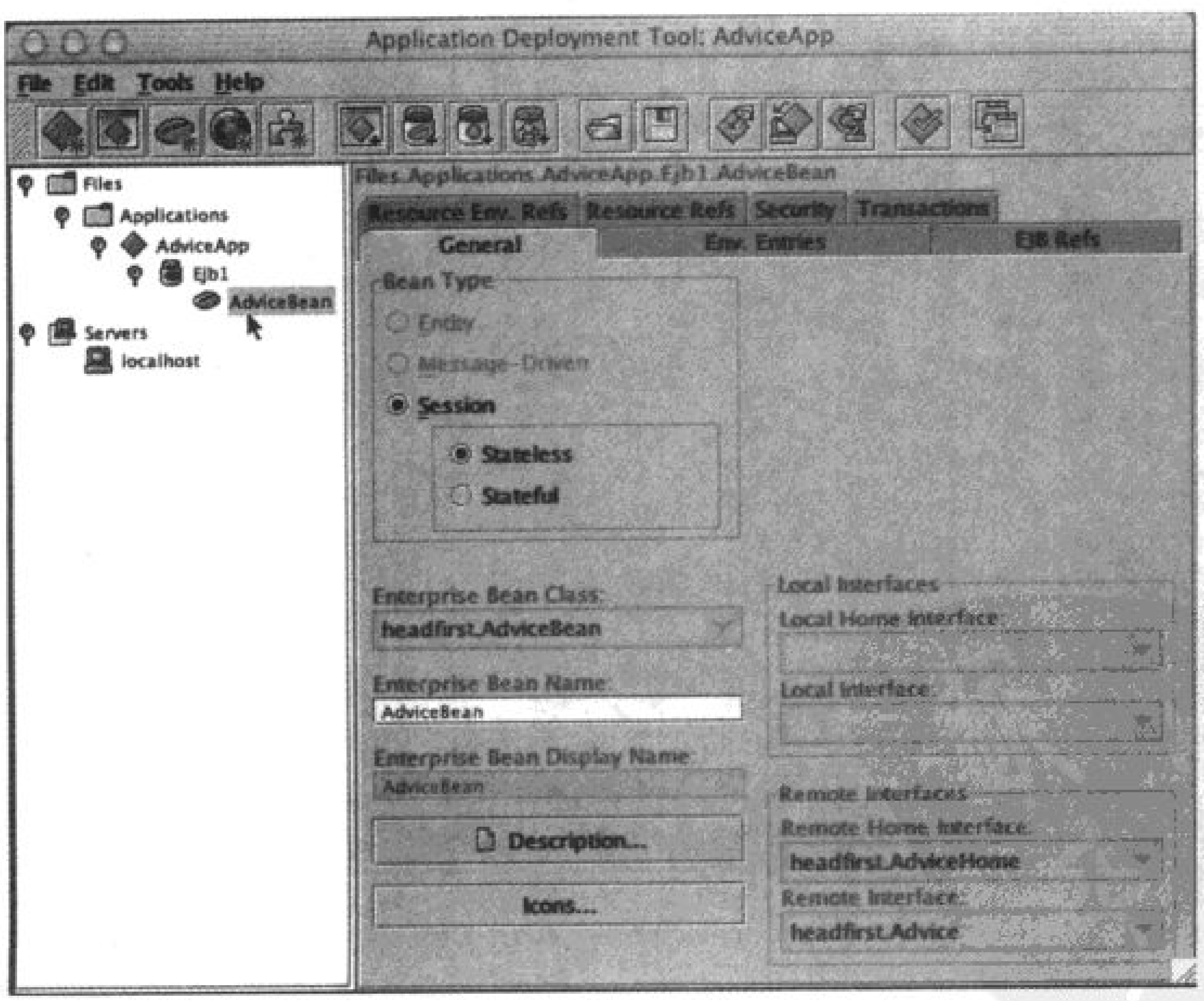
**点击 Finish**



# 再回到deploytool主界面……

一切都变了！AdviceApp图标现在展开了，显示出你的ejb-jar（名为“Ejb1”），这个ejb-jar也已展开，显示出可爱的bean图标（名为“AdviceBean”）。如果选择AdviceBean，就会看到一堆标签页，在此会显示出你在使用bean向导工具时做的选择。你选择的有些东西可以改变，但是有些是不能变的。例如，在General页中，你就不能改变为home接口、组件接口和bean指定的类文件。实际上可以看到，home接口、组件接口和bean所对应的下拉列表都置灰。不过，真是奇怪，在这里还是能把它从无状态改成有状态。

看看你的成果



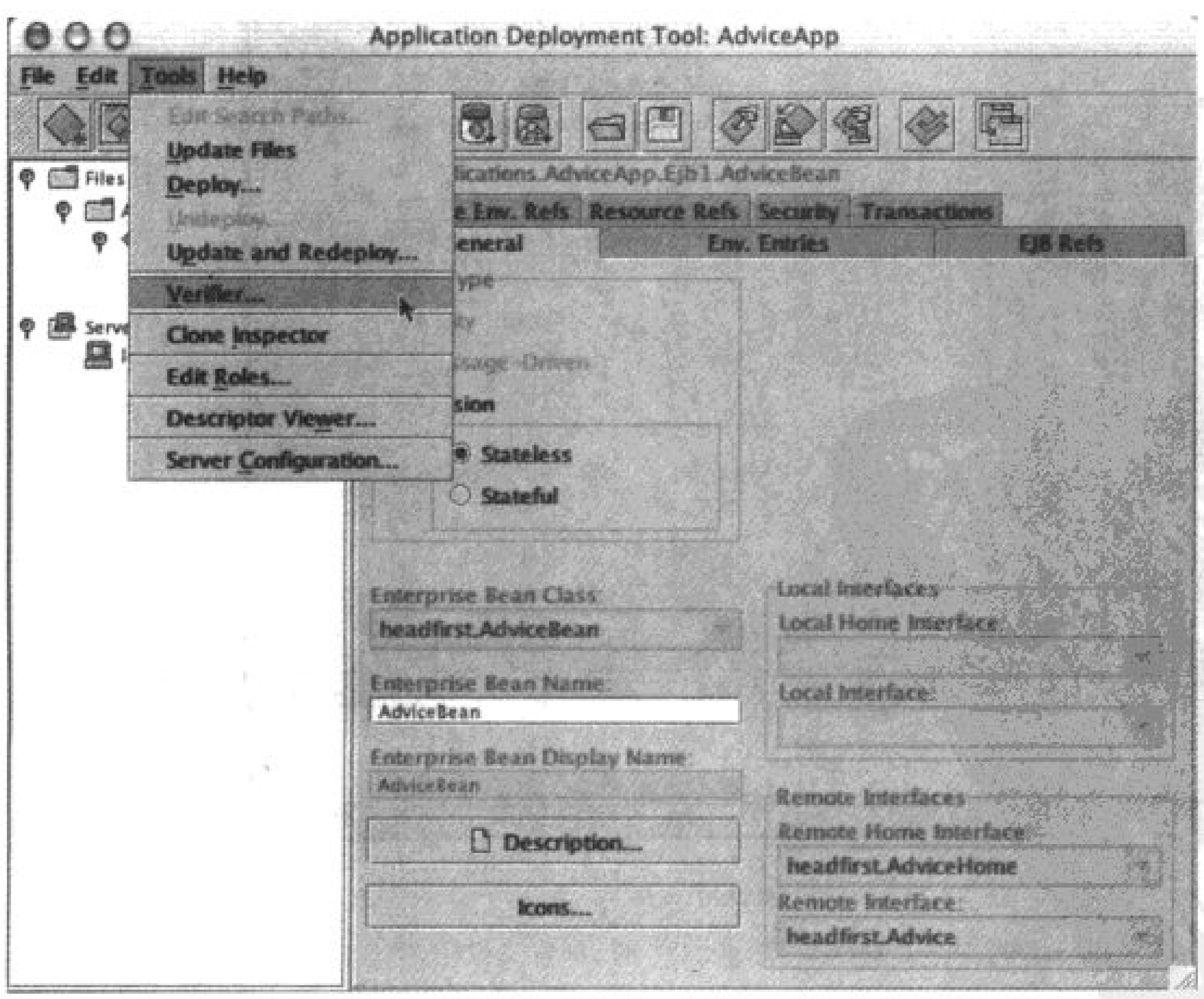
# 通过deploytool校验器运行你的bean

你已经知道你的bean类通过了编译，但是通过编译并不表示它一定遵循bean的法则。校验器要拿到你的jar，通过一组测试来运行，看它是否满足部署的最低需求。如果对EJB规范有更多认识，你就会了解到，校验器会测试你的bean代码（和部署描述文件），来看它是否遵循规范。例如，无状态会话bean的home接口必须声明一个无参数的create()方法，而且除此以外再没有任何其他方法。另外，bean类中的方法必须与组件接口中声明的方法相对应。如果组件接口是远程接口，就必须声明每个方法都抛出RemoteException异常。上面举的这些例子并不要求你记住；这只是让你对校验器做些什么工作有一个大致的认识。

点击Ejb1图标（小jar），让它加亮显示

选择 **Tools ▶ Verifier...**

坐在一边等着吧

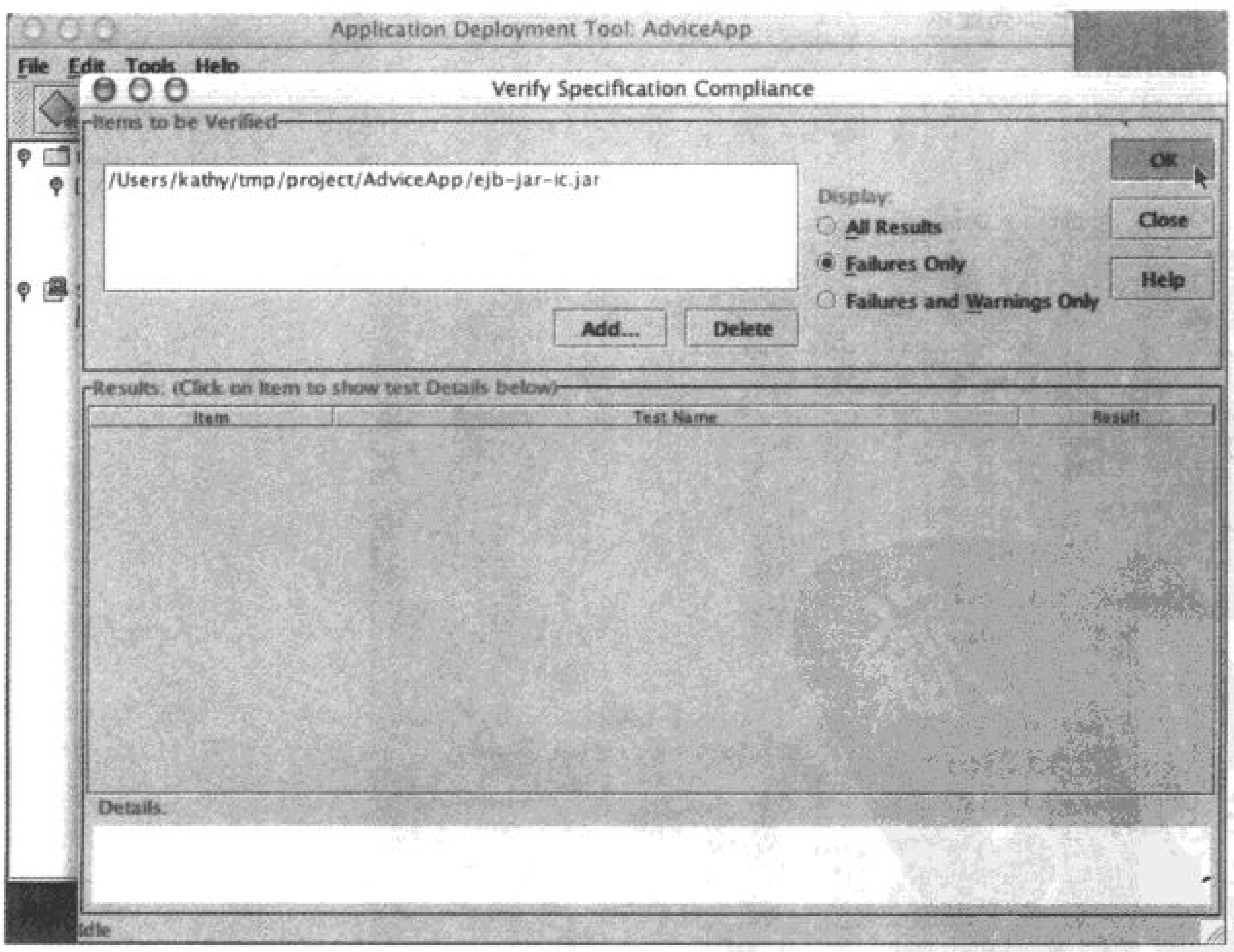


# 闭上眼睛，点击OK

校验界面把JAR显示为ejb-jar-ic.jar，放在一个tmp目录中，不过这只是一个临时位置，RI只是在你准备好部署之前把ejb-jar保存在这里。部署时，或者从应用删除这个bean时，这个tmp文件（和JAR）就会消失。

（至少一般认为是这样。有时，你可能必须自己找到这个目录，并将其删除。请跟我说一遍：deploytool是免费的。deploytool不是一个成品工具。我没有为这个工具交一分钱。我要学习怎么利用它的长处，而不要对它的不足之处太过苛责。）

**选择Failures Only 单选按钮**
**点击 OK**



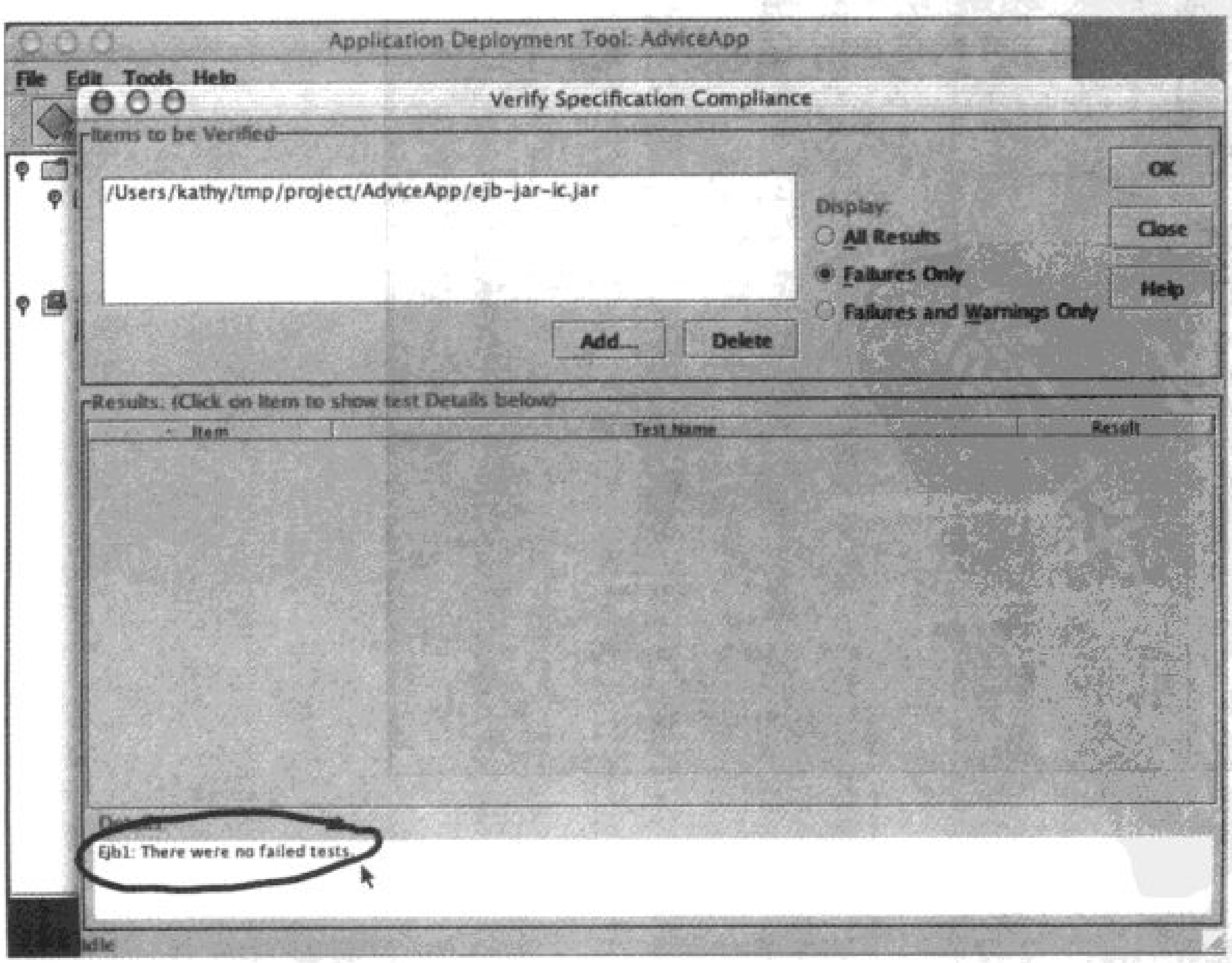
# 哇!测试都通过了

如果一切都校验通过，你会在校验器窗口底部的Details框里看到一条小消息。如果检验出错误，可以在Results框看到究竟是哪些错误。点击Results框里的一条“错误”消息，就能了解这个错误更详细的信息，明确到底哪里出了问题。如果你看到大篇大篇的错误也不要惊惶失措；通常只修改其中的一个，其他的也就没有了。校验器错误消息与编译器错误消息不同，编译器错误消息的帮助就像是VCR说明书一样，很难找出真正的问题所在，而校验器错误消息则相当明确，通常可以准确地得出问题出在哪里。

有时，通过点击某个标签页做点修改，就可以修正deploytool主窗口里的问题。例如，如果你忘了指定一个实体bean方法的事务属性，可以点击deploytool窗口中的Transactions（事务）标签页，设置这个属性，而不必再把bean向导工具运行一遍。

不过，如果是你的具体类文件有问题，就必须先修改（源文件），重新编译，再更新你的ejb-jar*。或者，如果问题出自你在bean向导中做了一个设置，而这个设置是不能修改的（比如，工具要求你指定home接口时，你却选择了bean类，由于类文件的选择是不可修改的，这样你就没有机会把它修改为真正的home接口了），那就必须删除这个ejb-jar，重新通过bean向导工具再来一遍。

**你会很高兴地看到底下的消息，显示一切正常**
**点击 Close**



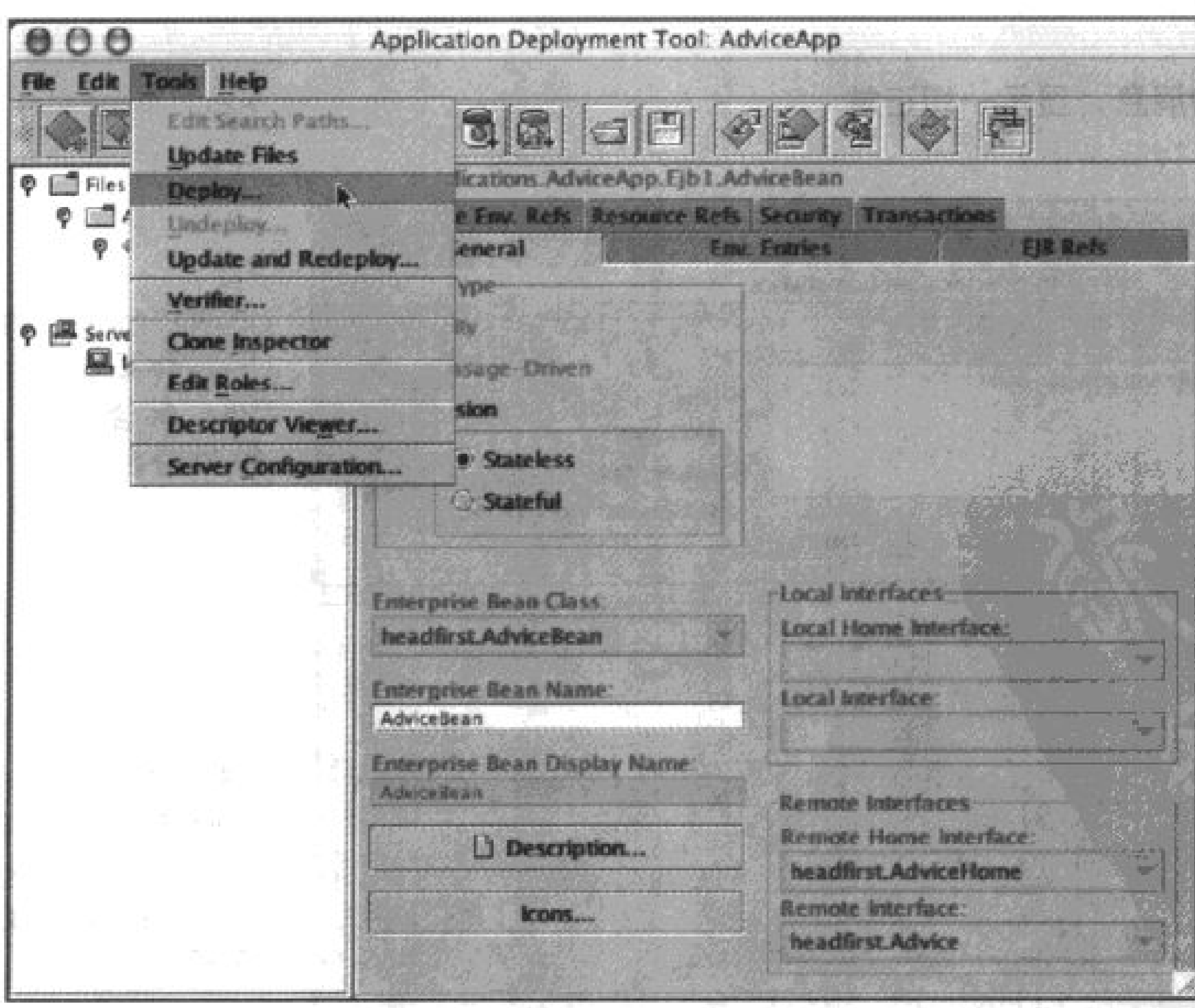

点击Close关闭窗口。如果点击了OK，就会再次运行校验器。

*要更新你的ejb-jar文件，先选择Tools，然后选择Update Files菜单项。

# 该部署了

一旦通过校验，下面就可以部署了。一般不会出什么问题，除非服务器里出了状况。如果是这样，只要关掉服务器和deploytool，再把它们重启，就万事大吉了，所有问题都会解决。是不是做法太强硬了？确实如此，但这种方法往往能奏效。有时你还必须采取更为严格的做法，要把服务器返回到它最初安装时的状态，不过幸运的是，有一个命令可以做这个工作。如果遭遇大面积失败，打开一个新终端，键入cleanup。这个脚本会关闭J2EE服务器，另外它还会清空这个机器上建立的所有日志、目录和文件。在此之后，可以尝试重启服务器，再重启deploytool，然后再次部署。如果你打开deploytool时没有看到你的应用，不要着急！只需选择File菜单下的Open，就像在其他应用中打开一个文档一样。对deploytool来说，确实可以把J2EE应用看作是一个文档。

**选择Tools ▸ Deploy……**



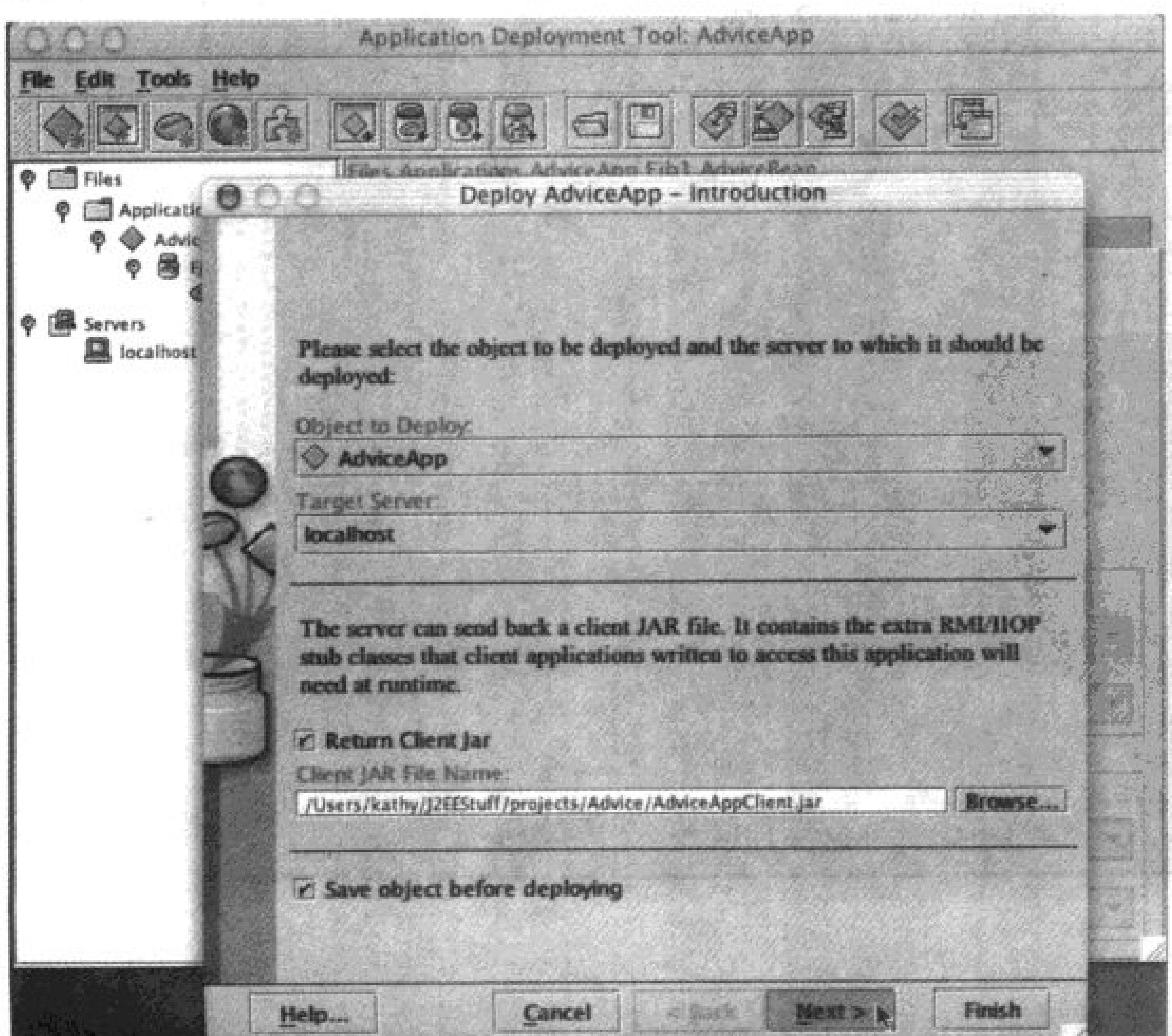
# 让它返回一个客户jar

服务器一旦启动，它会做许许多多的事情来准备你的bean。其中一个工作就是生成实现了两个接口（home接口和组件接口）的类文件。由于这两个接口是远程接口，所以服务器还要为这两个接口创建远程桩类（这个内容将在后面做更详细的介绍）。不错，客户需要两个接口和两个桩。你已经向客户提供了接口，因为这些接口已经创建。不过只有服务器才能建立桩类，而且如果没有这些桩类，客户是没办法工作的。如果真的没有桩类，客户可能只是老老实实地呆在那里，尽管已经编译好，但万事俱备只欠东风，它会等着你给它提供桩类，这样它才能实实在在地运行。

幸运的是，可以让RI deploytool提供一个客户jar，其中包括客户需要的所有东西（实际上，这个客户jar里的内容还有很多，不过由于这不是一个真正的成品环境，所以先不考虑这些）。

几乎所有EJB应用服务器都要创建桩类，所以你必须找出你的服务器把这些桩类放到哪里了，这样才能把桩类提供给客户。

**选择 Return Client Jar 复选框**（把它放到projects/advice 目录里）
**点击 Next >**

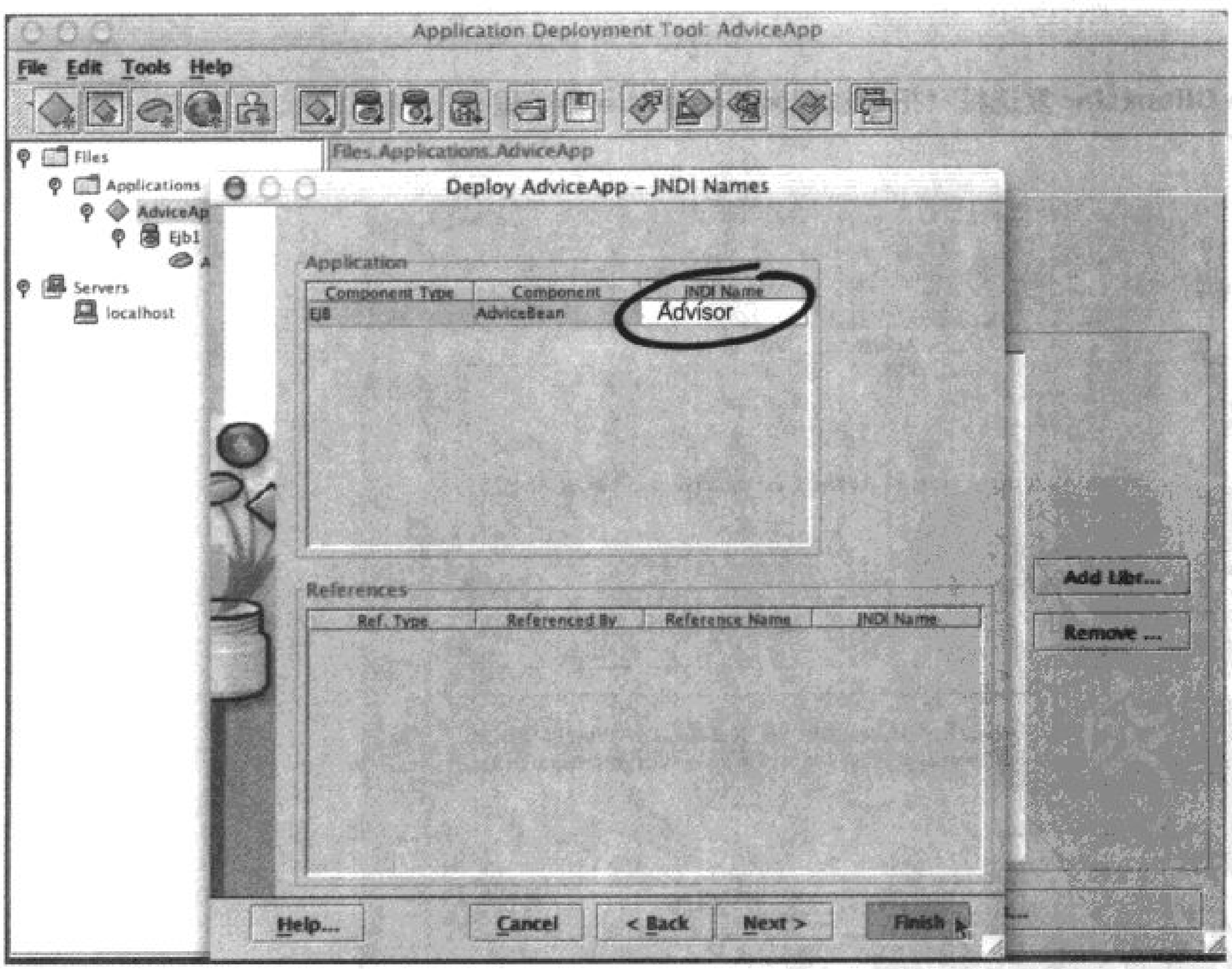


# 给它取个名字，以便客户查找

快要部署完了，最后一步是给bean提供一个JNDI名。客户会使用这个名字来得到bean的一个引用（当然，只是对客户来说这是一个bean，实际上并非如此，不过这些详细内容到下一章再做说明）。

bean的JNDI名只是你选择的一个逻辑名（或者是实际部署这个bean的任何人所指定的逻辑名）。不要求它与bean本身有什么对应。例如，我们可以把这个bean命名为Homer，实际上，对客户来说这比用一个有含义的名字可能更有意思。

键入JNDI名 **Advisor**
点击 **Finish**
深呼吸，等待，直到部署过程完成



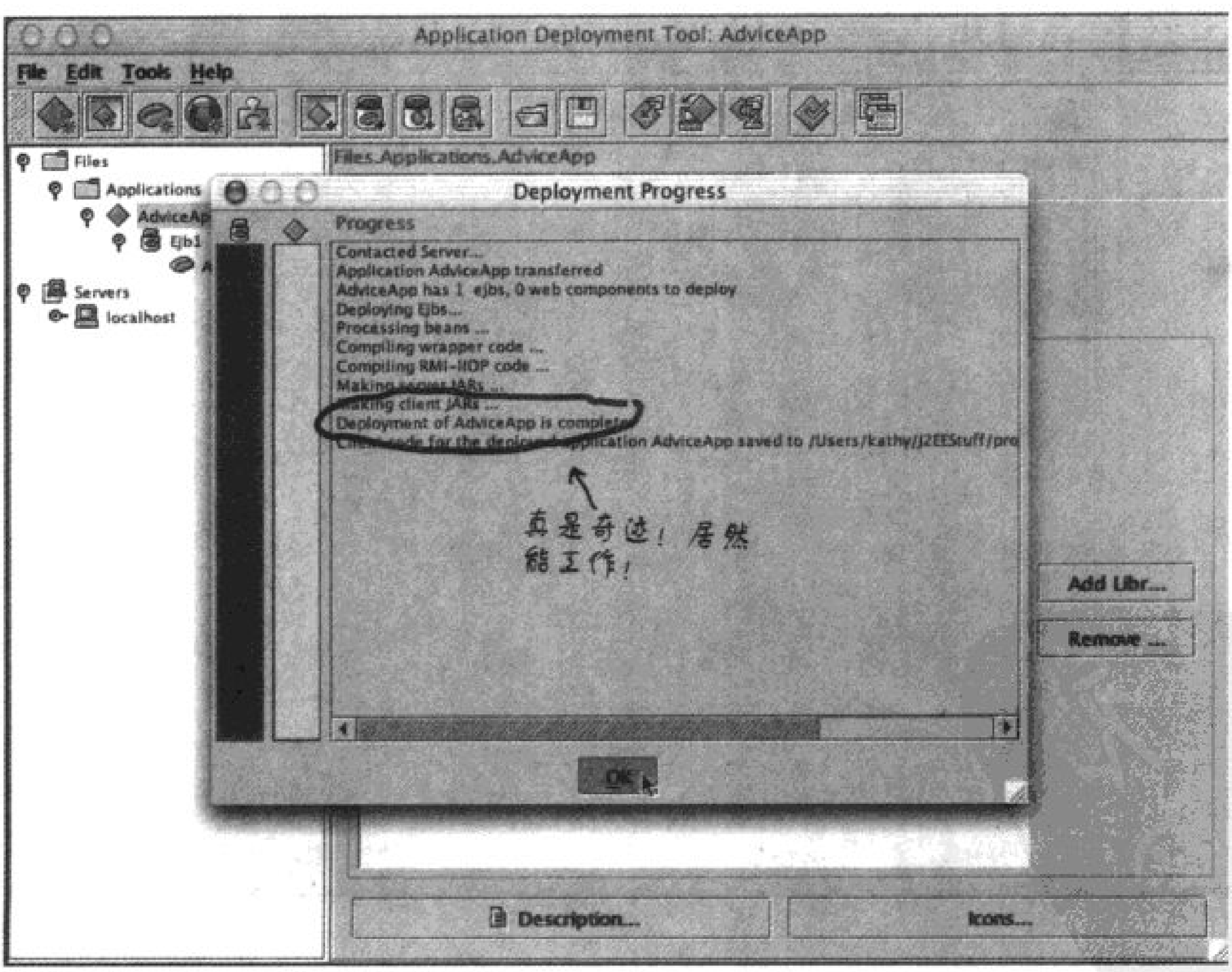
# 看着进度条前进，然后为自己祝贺吧

等待…… 等待…… 等待……

bean成功部署后，你会看到窗口中出现一行“Deployment of AdviceApp is complete（AdviceApp部署完成）”。现在，客户就可以访问bean了。

这下一切都好了！

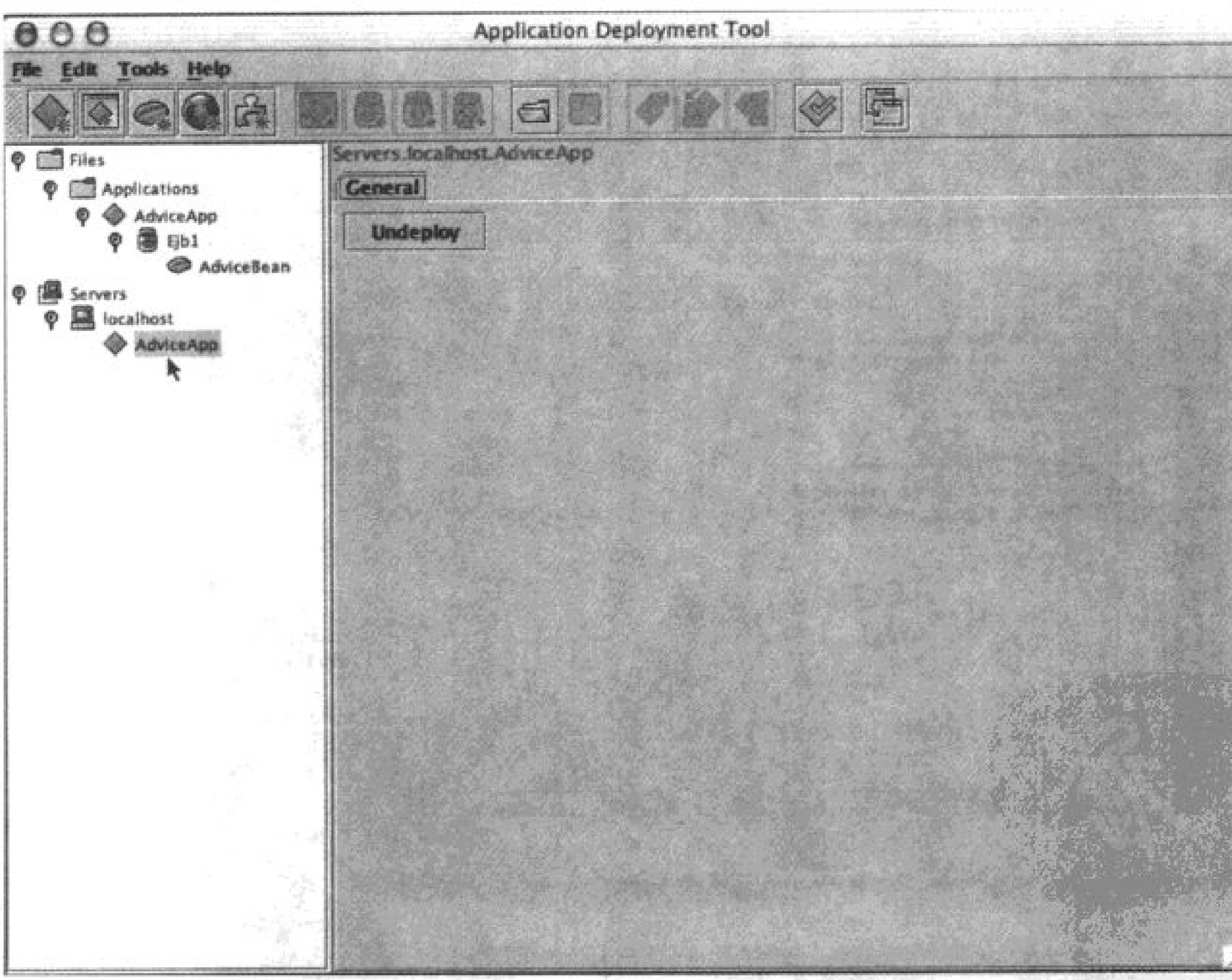
**点击 OK**



# 下面将可以看到，服务器里有了AdviceApp

在Servers图标的下面，你将会看到一个localhost图标，它表示在你启动deploytool之前启动的J2EE服务器。在这个localhost图标下面，现在可以看到你的AdviceApp已经部署好。另外还可以看到一个Undeploy按钮，顾名思义，可以用这个按钮取消部署。

**展开 Servers > localhost 图标，可以看到 AdviceApp**



# 现在只要一个客户就行了……

我们已经有了一个新部署的bean（在一个J2EE服务器中），但是如果没有客户，还不能测试这个bean。客户必须做五件事：

1. 得到JNDI InitialContext的一个引用（在“客户视图”一章会了解有关的更多内容）。
2. 使用InitialContext在bean（我们在部署时把这个bean取名为“Advisor”）的home接口上完成一个查找（lookup）。
3. 对查找返回的结果进行窄化（narrow）和强制类型转换（cast）（查找得到的是一个实现了AdviceHome接口的“东西”）。在“客户视图”一章将对窄化做更多的介绍。
4. 在home接口上调用create，返回组件接口的一个引用。
5. 在组件接口上调用getAdvice()（业务方法，还记得吗？我们为什么在这里，为什么要做前面的种种工作，不就是要做这个事情吗？），打印结果。

我们要用一个独立的Java程序作为客户。

在真实世界中，客户很可能是servlet或其他bean。

不管客户是什么类型，它要做的五件事都是一样的。

所以，尽管这里使用的是一个独立的Java程序，但你学到的东西同样适用于servlet客户。

如果客户是另外一个企业bean，代码会稍有差别，不过这五个步骤还是要做的。

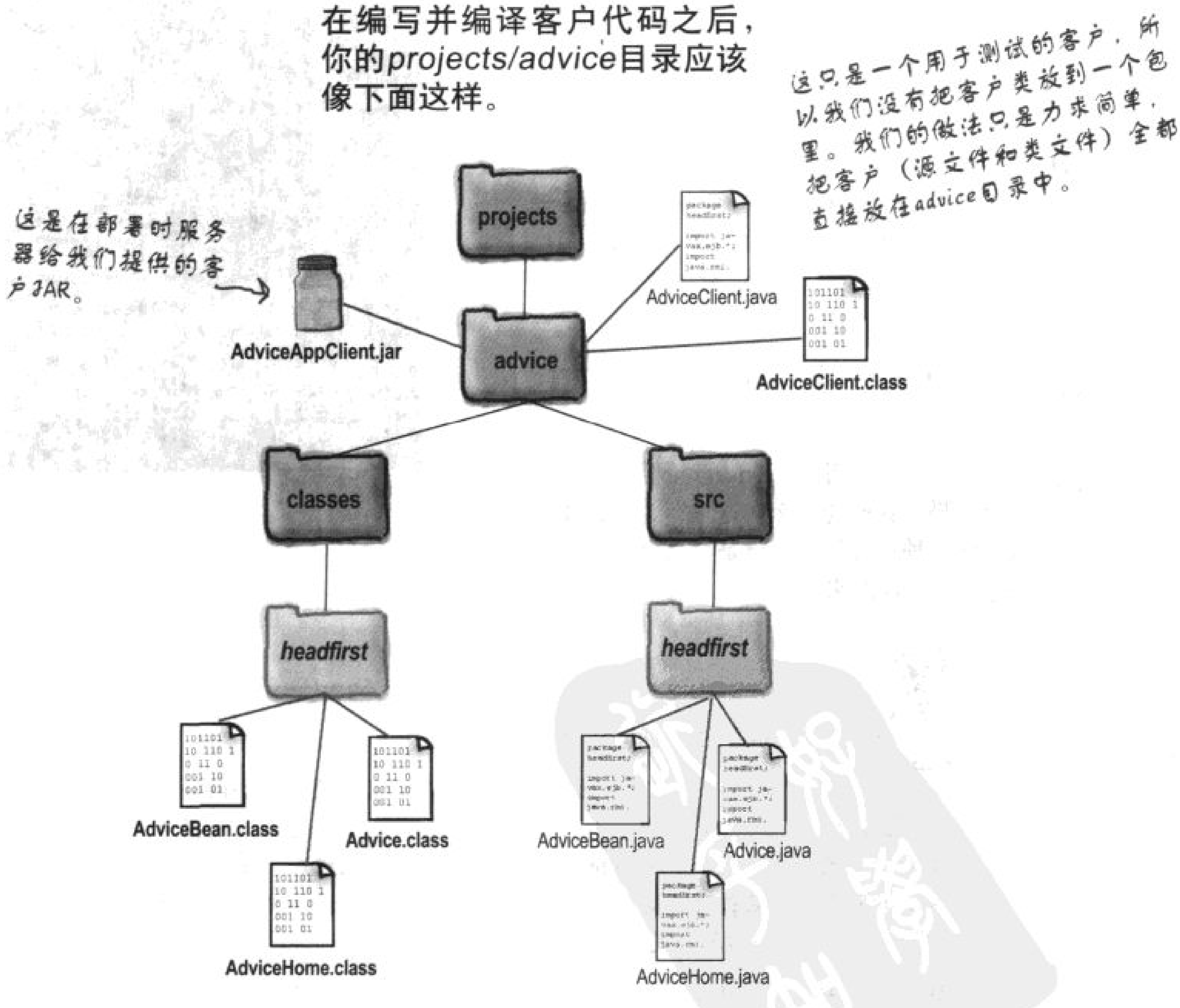
**Relax** 考试不会考到servlet或JSP。

考试要求你必须知道客户怎么得到和怎么使用一个bean（也就是上面的五个步骤），不过客户到底是哪一种类型，这倒没有什么关系。无论客户是一个servlet还是一个独立的Java程序，要得到一个bean的引用，并且最后在这个bean上调用方法，相应的客户代码几乎都是一样的。考试所关心的只是客户得到和使用bean的代码部分。

这个考试对servlet和JSP要求很低，你只要知道EJB规范不能保证支持servlet和JSP就足够了。servlet和JSP的支持不是EJB规范的任务，而要由J2EE规范来保证。

# 为客户组织工程目录

在编写并编译客户代码之后，你的*projects/advice*目录应该像下面这样。



## 客户代码（AdviceClient.java）

```
import javax.naming.*;
import java.rmi.*;
import javax.rmi.*;
import headfirst.*;
import javax.ejb.*;

public class AdviceClient {

   public static void main(String[] args) {
      new AdviceClient().go();
   }

   public void go() {
      try {
         Context ic = new InitialContext();
         Object o = ic.lookup( Advisor );

         AdviceHome home = (AdviceHome) PortableRemoteObject.narrow(o, AdviceHome.class);

         Advice advisor = home.create();
         System.out.println(advisor.getAdvice());
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

一堆导入（import）语句

InitialContext是JNDI命名服务的入口点，我们要在这里完成查找。

使用我们在部署时给定的JNDI名来查找Advice bean。

先按这样来，稍后会详细说明。

终于调用具体的业务方法了！这是在组件接口（Advice）中声明，并在bean类中实现的方法。

### 编译客户：

```
File Edit Window Help AllThisForJustThat
%cd ~/projects/advice

%javac -classpath  {$CLASSPATH}:AdviceAppClient.jar  AdviceClient.java
```

客户JAR文件

客户类

客户需要访问两个接口（Advice和AdviceHome）以及分别实现这些接口的两个桩类。它们都放在服务器建立的客户JAR中，如果没有它们，客户将无法编译。最简单的办法就是使用**-classpath**编译器标志，在编译时把这些接口和桩类（实际上是客户JAR文件）加到类路径中。

# 运行客户！

```
File Edit Window Help AllThisForJustThat
%cd ~/projects/advice

%java -cp {$CLASSPATH}:AdviceAppClient.jar   AdviceClient
```

在运行时，客户还需要访问两个接口（Advice和AdviceHome）和实现这些接口的两个桩类。它们都在服务器建立的客户JAR中，必须把它们增加到类路径中。最好的办法就是使用-**cp**编译器标志。

注意：就目前而言，客户肯定与服务器在同一个物理主机上。不过后面会看到怎么改变这种情况，也就是J2EE服务器在另外一台机器上的情况下如何运行客户。






# COFFEE CRAM 模拟测验

**1** 所有的EJB 2.0容器都必须实现或支持哪些特性？（选出所有正确的答案。）

- ❑ A. 一个GUI bean部署工具
- ❑ B. 对各种bean的同步调用
- ❑ C. 对各种bean的事务支持
- ❑ D. 各种bean的远程客户视图
- ❑ E. JNDI 1.2 命名空间

**2** EJB 2.0保证能支持哪些功能？（选出所有正确的答案。）

- ❑ A. 消息驱动bean的本地home接口
- ❑ B. 脏检测机制，以减少内存最低需求
- ❑ C. Run-as安全身份功能
- ❑ D. JDBC 2.0 扩展
- ❑ E. 会话bean故障恢复

**3** 下面哪些是EJB 2.0的特性？（选出所有正确的答案。）

- ❑ A. 可移植的查找方法查询语法
- ❑ B. 容器托管持久存储
- ❑ C. 会话bean的本地接口
- ❑ D. 基于XML的部署描述文
- ❑ E. 同步消息驱动bean

COFFEE CRAM

# 模拟测验答案

**1** 所有的EJB 2.0容器都必须实现或支持哪些特性？（选出所有正确的答案。） （规范：30）

- ☐ A. 一个GUI bean部署工具 — 不一定非得是GUI
- ☐ B. 对各种bean的同步调用 — MDB（消息驱动bean）不可能是同步的
- ☑ C. 对各种bean的事务支持
- ☐ D. 各种bean的远程客户视图 — MDB没有客户视图
- ☑ E. JNDI 1.2 命名空间

**2** EJB 2.0保证能支持哪些功能？（选出所有正确的答案。）

- ☐ A. 消息驱动bean的本地home接口 — MDB没有客户
- ☐ B. 脏检测机制，以减少内存最低需求 — 很好，但不能保证
- ☑ C. Run-as安全身份功能
- ☑ D. JDBC 2.0 扩展
- ☐ E. 会话bean故障恢复 — 可能有，但不能保证

**3** 下面哪些是EJB 2.0的特性？（选出所有正确的答案。） （规范：25—26）

- ☑ A. 可移植的查找方法查询语法
- ☑ B. 容器托管持久存储
- ☑ C. 会话bean的本地接口
- ☑ D. 基于XML的部署描述文件
- ☐ E. 同步消息驱动bean — MDB总是异步的

# 2 体系结构概述

# EJB体系结构




**EJB是基础架构。**组件是一些构建模块。利用EJB，除了简单的小应用外还可以构建非常大的应用。这包括各种规模的应用，从复杂的Victoria's Secret后端应用到简单的CERN文档处理系统，都可以利用EJB来构建。但是，作为一个体系结构要有这样高度的灵活性和可扩展性以及这么强大的功能，它肯定不是泛泛之辈。这种体系结构的基础往往是一个分布式编程模型，其中客户和服务器（甚至同一个应用的不同部分）都在网络上运行，但它们并不知道具体谁在哪里。不过，客户怎么找到一个bean呢？客户如何调用bean的方法呢？为什么存在不同类型的bean？这样的问题还有很多。

## 背景

这一章的目的是介绍背景，这些内容不在考试范围内。不过，这里讲的东西非常重要，可以这么说，考试的每个要求都和这一章有关，能不能达到考试要求就取决于你对这一章的理解。

不过，如果你想马上了解到底要考什么，请不要着急，从第3章开始，你就会看到许许多多的考试要求。等学到第6章，面对那么多的考试要求，你肯定会回想起这一章，想再重温现在的轻松日子："一个考试要求都没有……。"没有了才知道珍贵，等读到后面，你肯定会怀念这一章的，所以把握住现在的分分秒秒吧。

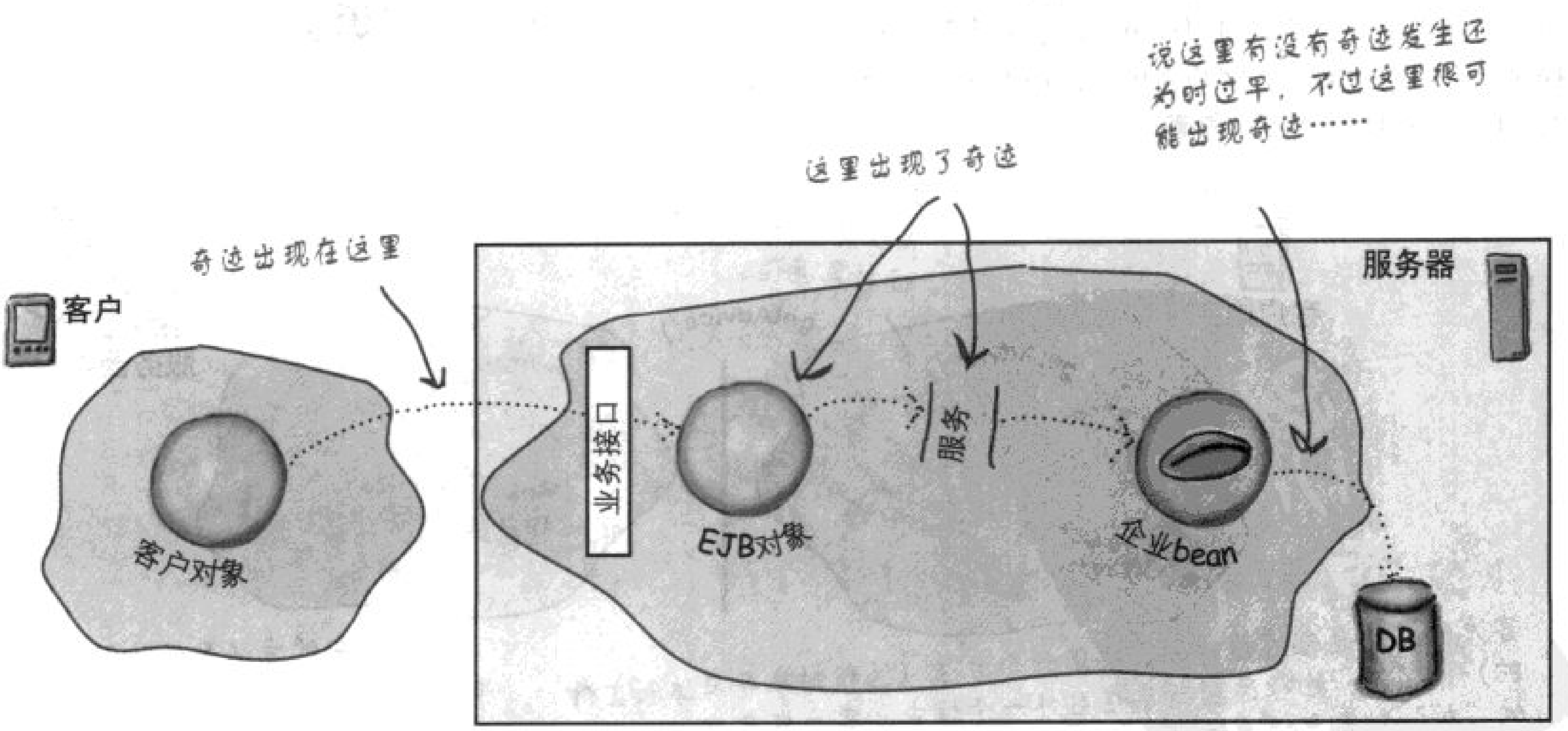
# 还记得这张图吧……

不过，这张图层次过高了，以至于我们无从下手。想想这张图里少了什么。比如，如果一个bean在另外一台机器上运行，客户怎么得到这个bean的引用呢？客户到底是怎样与bean通信的？客户调用bean的方法时，服务器可以介入其中，这到底是怎么回事？

在EJB的底层，采用了Java的一种分布式技术：远程方法调用（Remote Method Invocation，RMI）。尽管EJB对bean开发人员隐藏了RMI的一些复杂性，但是RMI仍然是存在的，而且除非你真的理解了RMI，否则对EJB的某些部分可能永远也搞不清楚。

所以，我们要从高层视图走下来，深入到EJB的内部中去，先来上堂RMI课。如果你很幸运，已经用过不少RMI，那你可以跳过RMI的介绍，直接去看后面有关EJB如何使用RMI的部分。不过，即便如此，还是应该至少大致翻一下，即使你已经是一位很有经验的EJB开发人员，如果想更轻松地学习这本书，更顺利地接受这本书后面用到的术语和示意图，单从这一点来说，也应该浏览一下这一章。

那好，再回到正题上来，这个图里少了些什么？先来看哪里会出现奇迹……




EJB体系结构的高层视图（漫画版）

# 建立一个远程方法调用

编写客户来访问一个bean时，这个客户可能是本地（local）客户，也可能是远程（remote）客户。本地客户是指，这个客户与它访问的bean在同一个JVM中运行。换句话说，bean和客户生存在同一个堆（heap）中。这个内容将在关于“客户视图”的一章做更多的介绍，目前需要记住，本地就表示在同一个堆/JVM中。尽管可能有本地客户，但是一般只对实体bean使用本地客户，而且只在非常特殊的情况下才会使用本地客户。

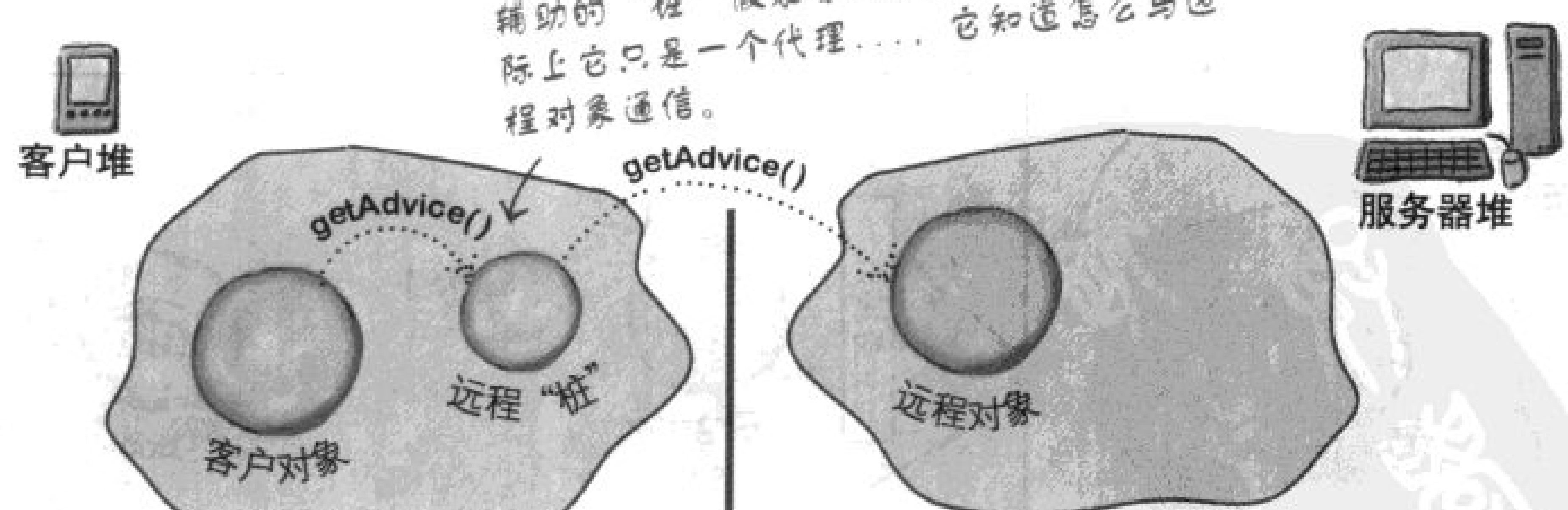
如果希望一个bean能够被外界使用，就可以使用一个远程客户。尽管应用中使用的一些bean相互之间会作为本地客户进行通信，但是大多数企业应用都有一个远程客户。（在这本书中，我们还会对此详细地进行分析。）

那么，如果两个对象分别在不同的堆/JVM中运行，前一个对象怎么在后一个对象引用上直接调用一个方法呢？从技术上讲，这是不可能的！Java引用中的内容（比特位）出了当前运行所在的JVM就没有任何意义了。如果你是一个对象，而且你有另一个对象的引用，那个对象就必须和你在同一个堆中。

Java RMI（远程方法调用）可以解决这个问题，它为客户提供了一个代理对象，这称为桩（stub），这个代理对象相当于客户和远程对象之间的中间人。客户在桩上调用方法，桩则负责完成与远程对象的底层通信（socket和流）。

**利用RMI，你的客户对象会表现得好像在做一个远程方法调用。但实际上，它只是在一个“代理”对象上调用方法，这个代理对象在客户所在的同一个堆中运行。这个代理称为“桩”，它会处理所有底层的网络socket和流。**

辅助的“桩”假装自己是远程对象，不过实际上它只是一个代理……它知道怎么与远程对象通信。




客户的目的是在远程对象上调用一个方法（远程对象是具体做工作的）。不过，由于远程对象在另外一个堆中，客户只是调用一个本地“桩”对象上的方法（这个桩对象再通过网络发送信息）。

（译者注：图中称桩为“远程桩”，这表示它是远程对象的桩，但这个桩与客户对象同在一个堆中，所以相对于客户来说是本地的，因此在对客户的说明中提到了“本地桩”，理解的时候要区别清楚。）

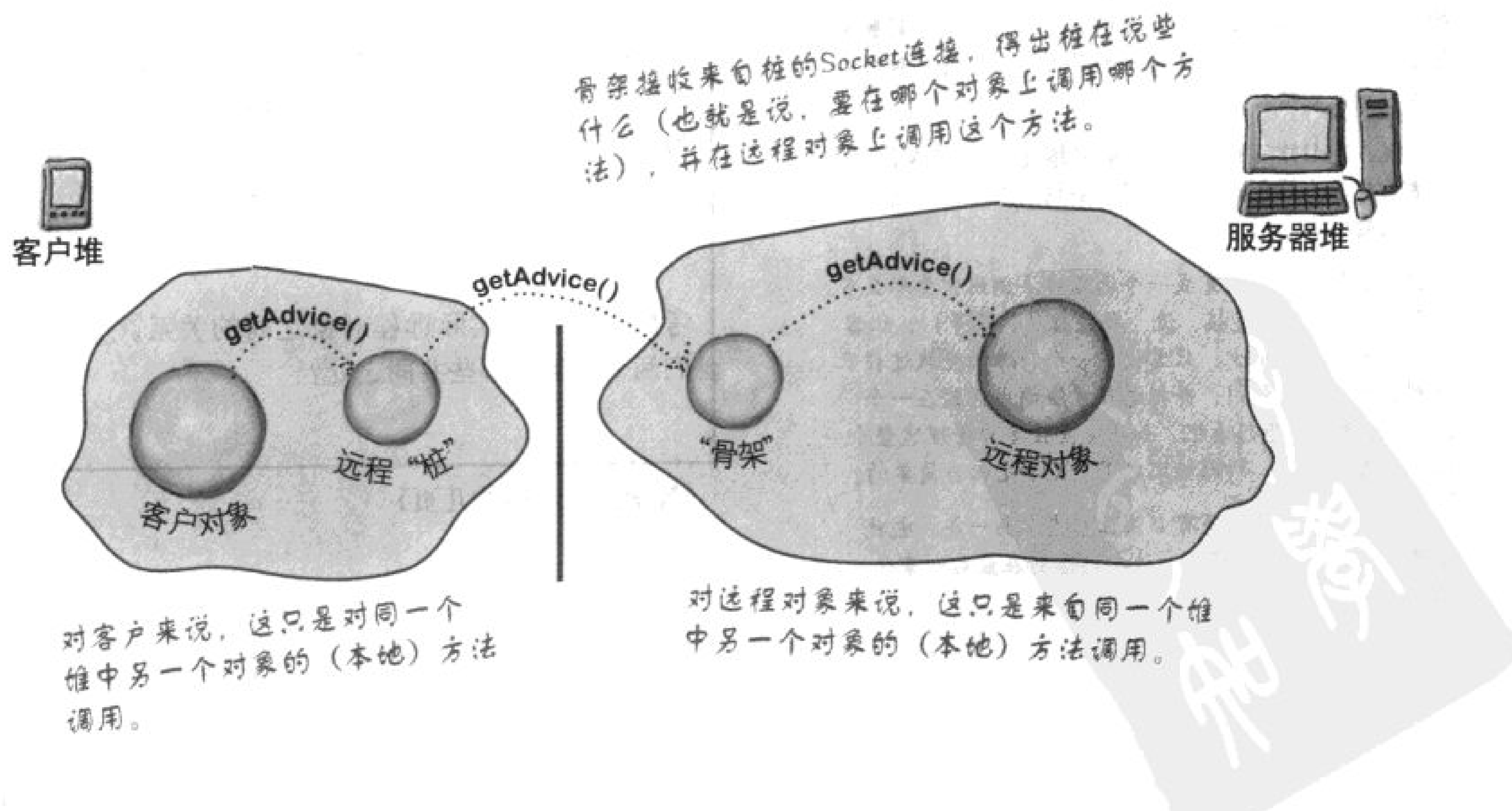
远程对象中的方法是具体做工作的（做什么工作不一定，比如checkCredit()、calculatePi()等等）。

# 服务器端也有一个"辅助"对象……

远程对象中有客户想调用的方法。但是，当桩建立与服务器的网络连接时，服务器上必须有人来接收到来流中的信息，并把它转换为对远程对象的一个方法调用。可以把网络通信代码放在你的远程对象中，但是这就有违RMI的主旨了，RMI就是想让客户能很容易地调用网络上一个对象的方法，就好像在调用同一个堆中某个对象的方法一样。RMI的目标是提高网络的透明性。换句话说，也就是不同机器上的对象对开发人员来说都应该是透明的，这对你来说则意味着……代码更少，而且更简单。

所以，本着这个目标，RMI还会负责方法调用的服务器端。服务器端接受socket连接的对象称为骨架（skeleton）。它与客户端的桩对应。在RMI的早期版本中，对应每个桩，都有一个与之匹配的骨架对象。不过，如今这可不一定成立了。必须在服务器端以某种方式完成骨架的功能，不过具体的骨架对象是可选的。我们不再深入讨论这些细节，因为如果是基于EJB，有没有骨架对象对我们来说没有什么差别。容器如何实现其骨架行为，这由开发商决定。

我们关心的只是：服务器端有这么一个"东西"，桩知道怎么与它通信，另外它也知道怎么解释来自桩的消息，并在远程对象上调用一个方法。



## there are no Dumb Questions

**问：** **怎么才能得到“网络透明性”？客户调用远程方法的时候，如果网络或服务器瘫了会怎么样？如果不涉及网络，客户对象只是对同一个堆中的另一个对象做普通的方法调用，相比之下，现在看上去好像更多的地方都有可能会出错。**

**答：** 是这样的。显然你已经很了解，“网络透明性”并不像人们传得那么神乎其神，它同时也是一个糟糕的想法。当然了，本地方法调用不会出错的地方，远程方法调用却有可能失败，而且客户必须准备应付这些情况！

这也说明了为什么Java RMI中所有远程方法都必须声明一个java.rmi.RemoteException异常，这是一个受查异常。所谓受查异常，意味着客户必须处理或声明这个异常。换句话说，客户要想假装这个方法调用不是远程的，这是办不到的。

不过，稍等，还不只这些，甚至在客户第一次获得远程对象引用的时候也要做些特殊的事情。这个引用到底是什么？它实际上不是远程对象的引用，而是远程对象的代理（也就是桩）的引用。

所以说，RMI并不能给你真正的网络透明性。RMI的设计者希望客户能认识到，远程方法调用中可能会发生严重的问题。

不过，虽然可以看到完成一个远程方法调用需要那么多东西（网络Socket连接、流、参数打包，等等），但客户要做的事情却很少：只需使用一个特殊的查找过程来得到远程对象的引用，并把远程方法调用包装在一个try/catch里。你想想看吧，如果完全让客户管理这整个过程该有多可怕，所以相比之下，RMI是相当简单的。

（对客户来说，甚至还有办法让它更容易一些，也就是使用一个EJB设计模式，这个内容将在最后一章介绍。）

**问：** **要我负责构建桩和骨架吗？桩怎么知道我的远程对象有些什么方法呢？再说了，客户又怎么知道我的远程对象有些什么方法呢？**

**答：** 不用，你不需要自己来建桩和骨架。对于普通的RMI，使用RMI编译器（rmic）就可以生成桩和骨架。不过，对于后两个问题……在我们详细说明之前，你先自己考虑一下。

### BRAIN POWER

在Java中，要想告诉客户能调用哪些方法，什么办法最好？也就是说，你怎么把公共方法展现给别人？

考虑一下桩与实际远程对象之间的关系。它们两个必须有哪些共同之处？

（答案请见随后的几页）

# 关于参数和返回值

远程方法调用与本地方法调用基本上是一样的，只不过远程方法需要声明RemoteException异常。如果不能传递参数或者不能得到一个返回值，那这个方法调用还有什么意义呢？你也可以用古老的远程过程调用（RPC）*，这要算你的父辈用的办法了。

从这里可以看出桩和骨架的一个重要任务（也可以不是骨架，而是其他的一个什么东西，只要能做骨架的工作就行），这就是要对网络上传递的值进行包装（packing）和解包（unpacking）。

要记住，客户实际上调用的是桩上的一个方法，而桩对客户来说是本地的（也就是说，它们在同一个堆中）。所以，在客户看来，发送方法参数并没有什么特别的地方。所有麻烦的事情都是桩来做的。桩必须把参数包装起来（这要通过一个称为编组的过程），然后通过与服务器的Socket连接把打包好的参数发送到输出流中。

服务器上的骨架（或者骨架之类的东西）必须处理从桩发出的流，解开包得到参数，明确该怎么做（例如，在哪个对象上调用哪个方法），然后基于这些参数在远程对象上调用这个方法（这是一个本地调用）。

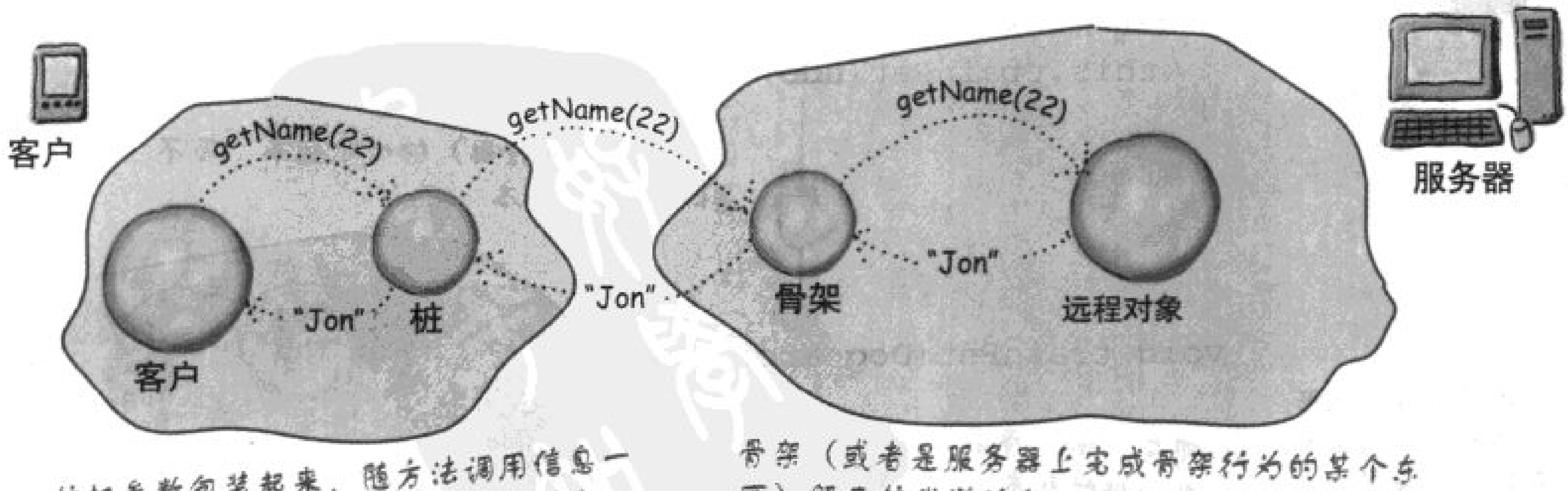
接下来再把上面的过程反过来走一遍！骨架把返回值包装起来，传递给桩，桩则解开包得到返回值，并把它们作为普通的返回值提供给客户。不过，要想发送参数和返回值，这些参数和返回值只能是以下的某种类型：基本类型、Serializable（可串行化）对象，由基本类型或Serializable对象组成的数组或集合，或者是一个Remote（远程）对象。

桩和骨架会完成一个完整的往返通信。它们都要负责对网络上传递的值进行包装和解包。

不过，如果参数和返回值本身不可传递，这种做法就不可行。

*可传递（Shippable）的值必须是以下几种类型：*

- 基本类型
- Serializable对象
- 由基本类型或Serializable对象组成的数组或集合
- Remote（远程）对象




桩把参数包装起来，随方法调用信息一同发送出去，然后解开骨架返回的包，得到提供给客户的值。

骨架（或者是服务器上完成骨架行为的某个东西）解开桩发送的包，得到提供给远程对象方法调用的方法参数，然后将返回值打包，并传送。

*RPC代表远程过程调用（Remote Procedure Call）。这个技术很难用。

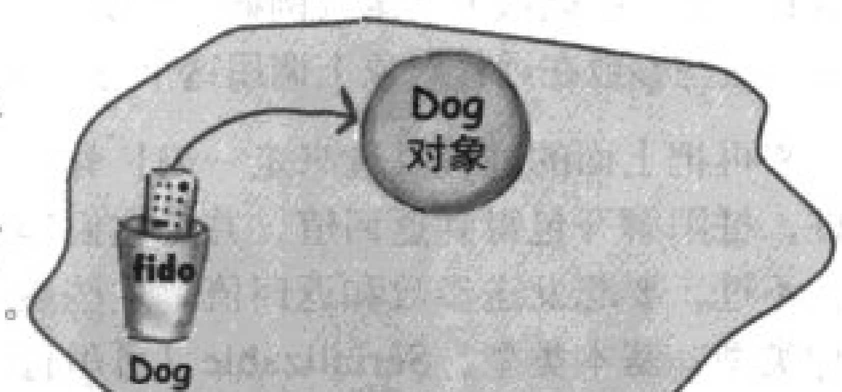
等一等…… Java传递对象时，传递的是对象引用的一个副本，而不是传递对象本身的副本。那远程方法调用是怎么做到的呢？要知道，这个引用如果到了另外的一个堆上，那就没有任何意义了……



**在正常的本地方法调用中，Java按值传递对象引用。换句话说，它会复制引用变量中的比特位。**

**对象本身是不会传递的。**

我们把本地Java引用看作是同一个堆上的一个对象的遥控器。我们可以用这样一个遥控器对实际的对象"按按钮"（也就是调用方法）。



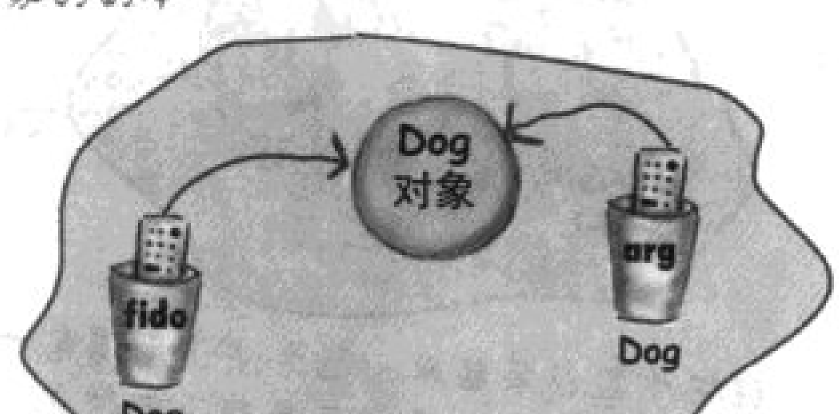

```
void go() {
  Dog fido = new Dog();
  this.trainPet(fido);
}
```

这里传递的到底是什么??

这是引用（遥控器）的一个副本，而不是Dog对象的副本

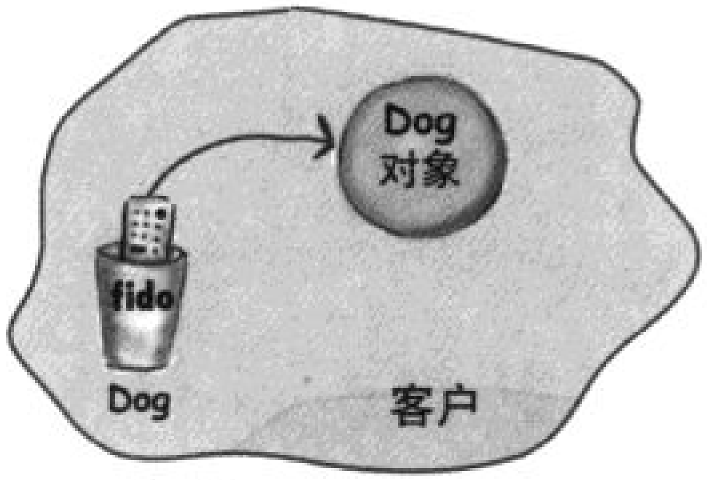
```
void trainPet(Dog arg)  { }
```

现在"arg"参数和"fido"变量都是一样的副本。它们都引用同一个Dog对象。




当然，我们很清楚这些东西你都知道，正是如此"我们才会在同一条船上"，不过你还应当了解这里面的思想。

# 向一个远程方法传递对象时，到底传递的是什么？




假设**客户**代码是这样的：

```
try {

  Dog fido = new Dog();

  remoteStub.trainPet(fido);

} catch(RemoteException ex){
    ex.printStackTrace();
}
```

现在，这里到底传递的是什么？？




不是引用值！这是实际Dog对象的一个可串行化副本。

下面是远程方法：

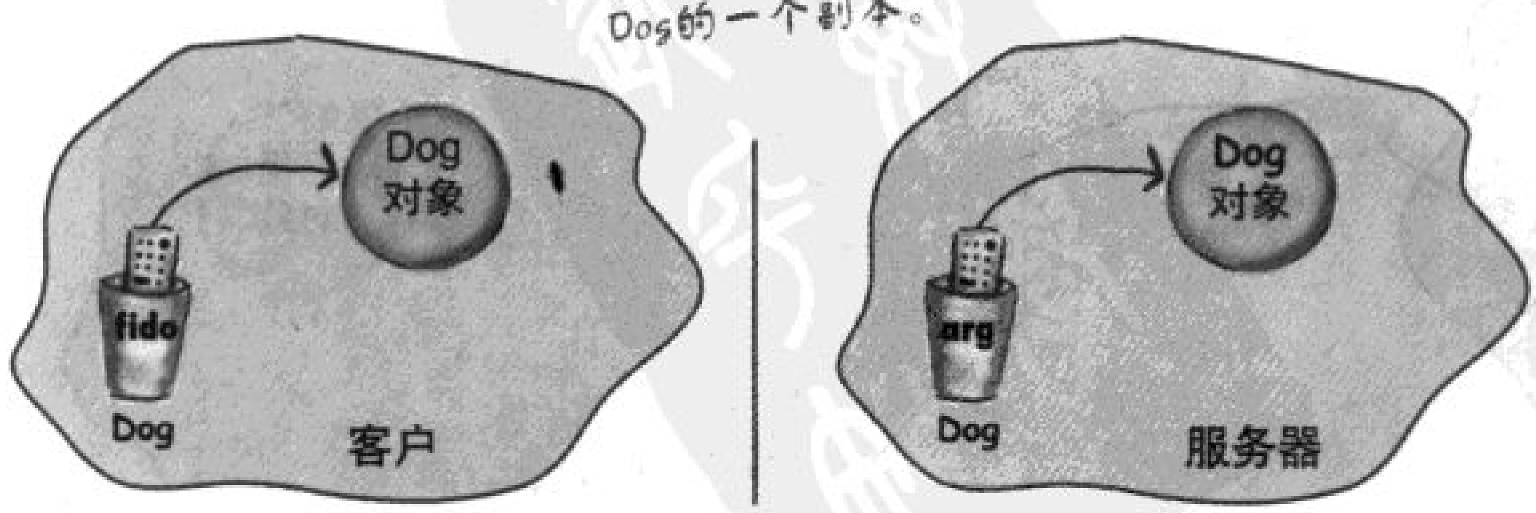
```
void trainPet(Dog arg) { }
```

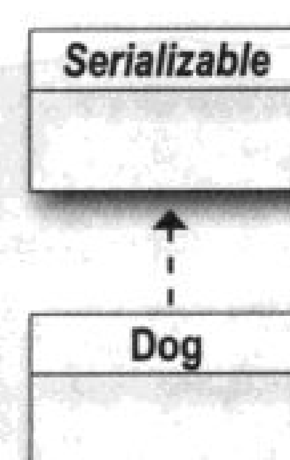
如果你的远程方法有一个对象类型的参数，这个参数会作为对象本身的一个完整副本来传递！

对于远程调用，Java按对象副本来传递对象，而不是按引用副本传递。

对象的串行化副本会传递到远程对象。

服务器上的Dog是客户中Dog的一个副本。

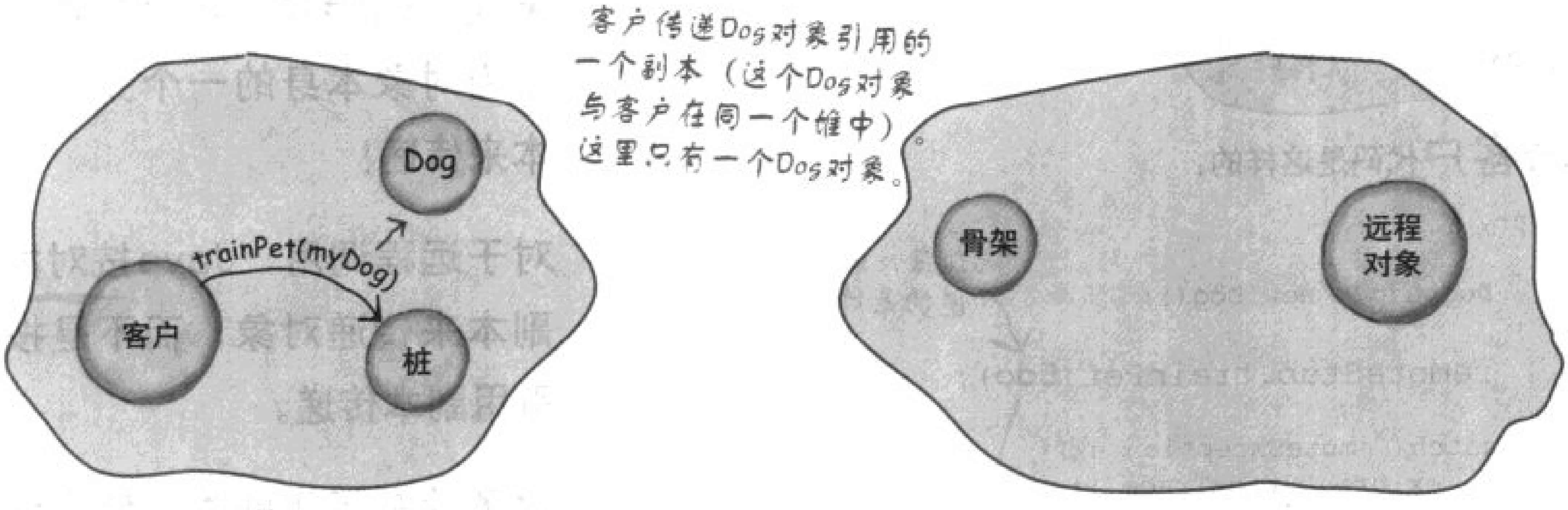


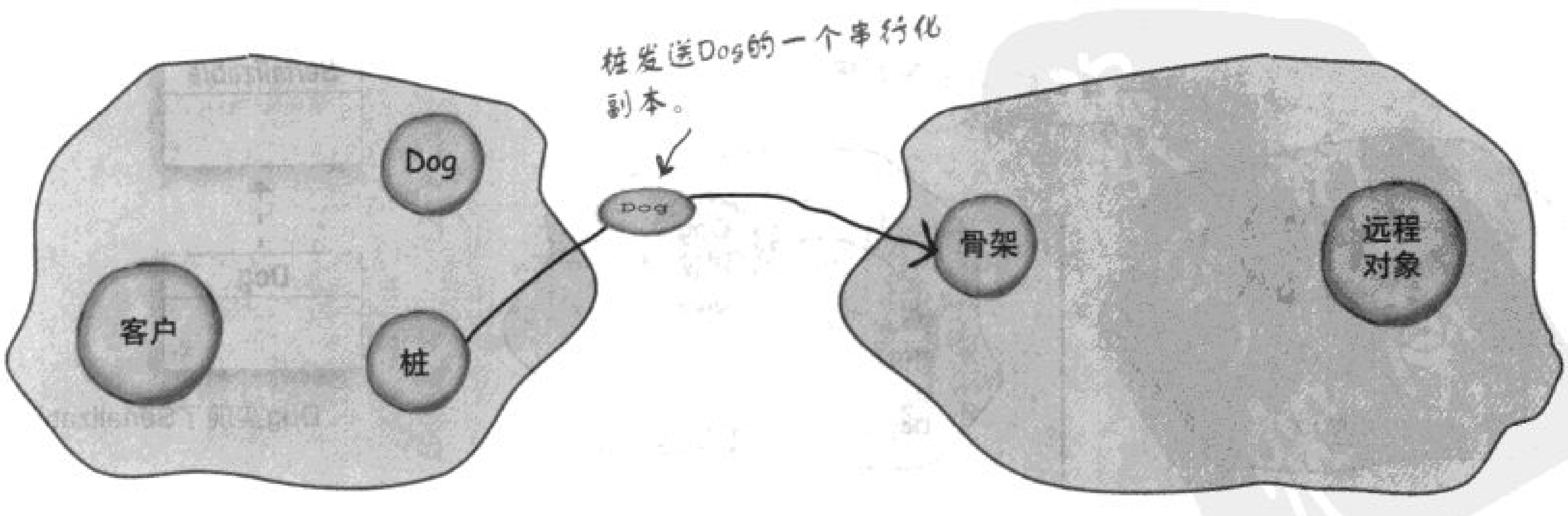

Dog实现了Serializable

# 获得客户传递给服务器的对象参数

① 客户在桩上调用trainPet(myDog)，并传递Dog对象的引用的一个副本。

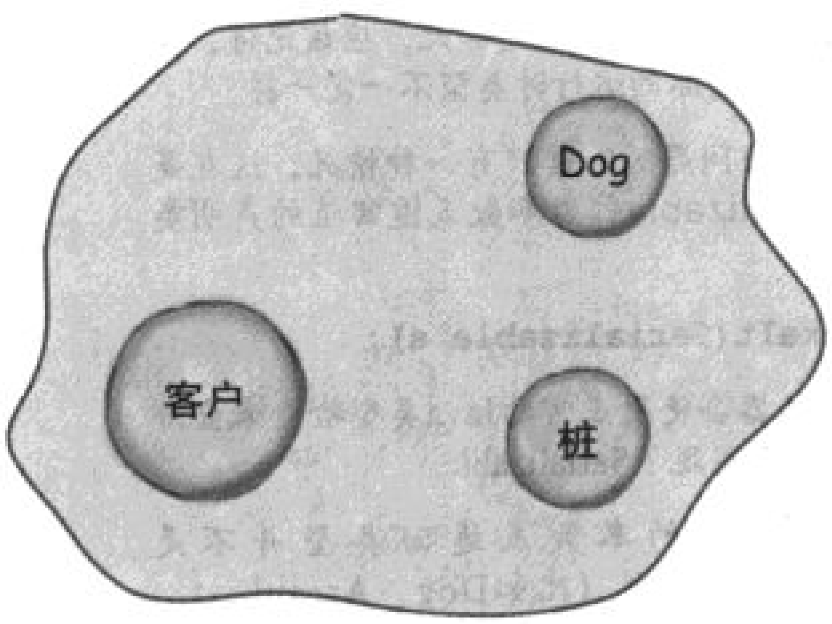


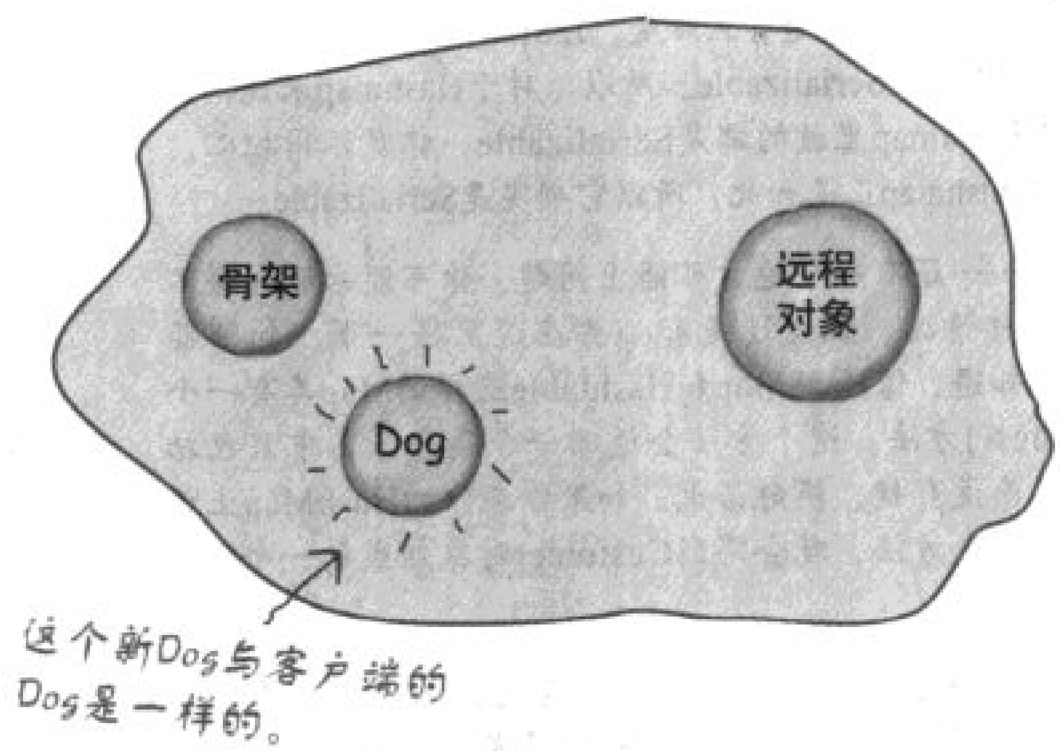

② 桩建立这个对象的一个串行化副本，并通过网络把这个副本发送给骨架。



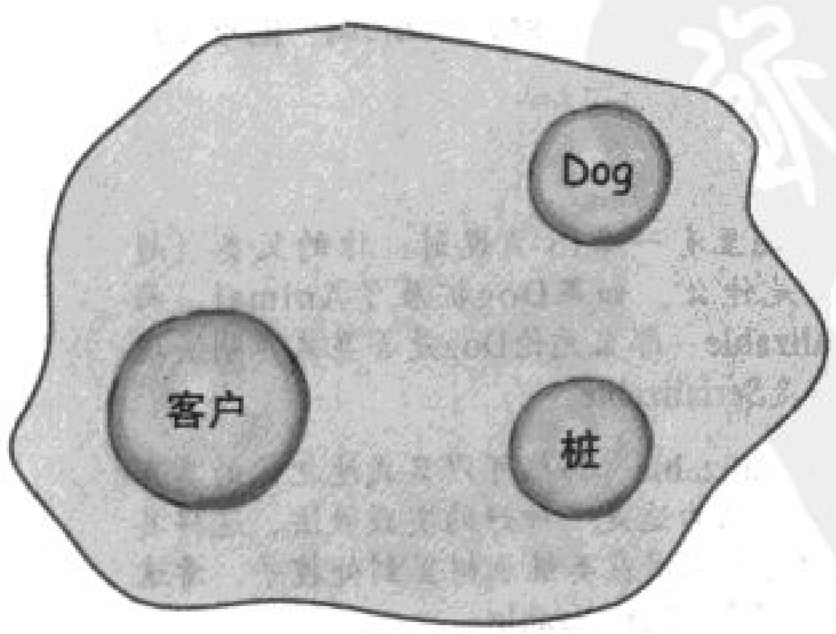
## 在服务器上解开（逆串行化）对象

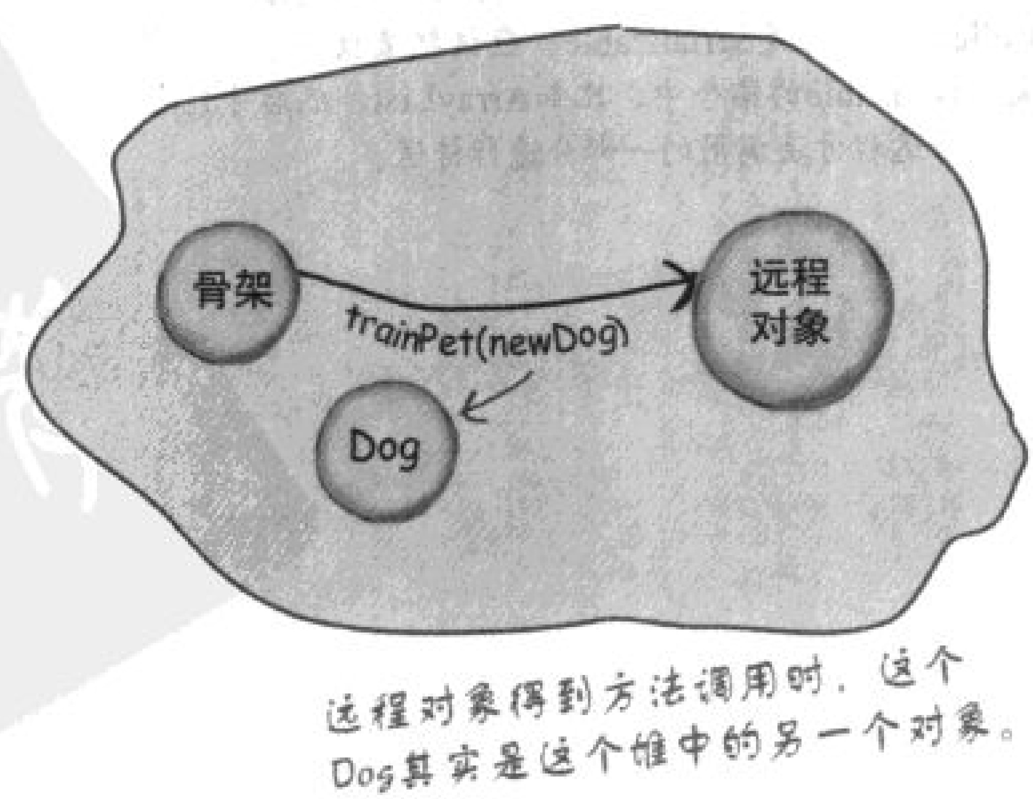
③ 骨架对传递来的参数逆串行化，在远程对象所在的堆创建一个新的Dog对象。




④ 骨架调用远程对象上的方法，并传递新Dog对象的一个普通Java引用。



## there are no Dumb Questions

**问：** **我有一个HashMap，里面装的是 Serializable Customer对象，并配有String键。这个HashMap本身是Serializable吗？我需要考虑这个问题吗？**

**答：** 这个问题有点难度。J2SE API中的所有Collection实现都是Serializable。所以，对于Hashmap来说，只要在Hashmap里放的都是Serializable，你就不用担心。你的Hashmap正是如此，所以它确实是Serializable。

不过……还有一个地方可能出问题。你可能永远也碰不到这种情况，但是还是有必要在这里提一下。你可能已经知道，像HashMap和Hashtable这样的Map类有一个values()方法，这个方法会返回一个集合，其中只包括值，而没有键。换句话说，如果你在你的HashMap上调用了这个方法，就会得到Customer对象组成的一个Collection。

但是这个集合是什么类型呢？这就有问题了。你不知道它的类型是什么。你知道的只是它实现了Collection接口。但是，要想知道values()返回的Collection是不是Serializable，这还远远不够！也就是说，你拿到的Collection可能是这样的：尽管这个集合中的元素都是Serializable对象，但这个集合本身却不是Serializable！

重点是：不要指望映射（map）集合的values()方法返回的Collection一定是Serializable。应该把值放在你相信肯定是Serializable的集合中，比如ArrayList，然后才能把它们作为远程方法调用的一部分进行传送。

**问：** **如果传给一个远程方法的类型不是Serializable，这是一个编译时错误还是一个运行时错误呢？**

**答：** 运行时错误！一般来讲都是如此。应该记得，声明的参数或返回类型与实际的运行时类型不一定一样。

什么时候会在编译时出问题呢？这只有一种情况，这就是远程方法正是用Serializable作为参数或返回值的声明类型：

```
public void takeIt(Serializable s);
```

在这种情况下，编译器会使用正常的Java类型检查来查看所传递的声明类型是否实现了Serializable。

大多数情况下，声明的参数或返回类型并不是Serializable，而是其他类型（比如Dog、ArrayList、String等等），所以Java在真正完成串行化之前并不知道运行时对象是不是Serializable，这个时候就会抛出一个异常。

对于集合和数组，如果其中的某个对象不是Serializable，那整个串行化都会失败！

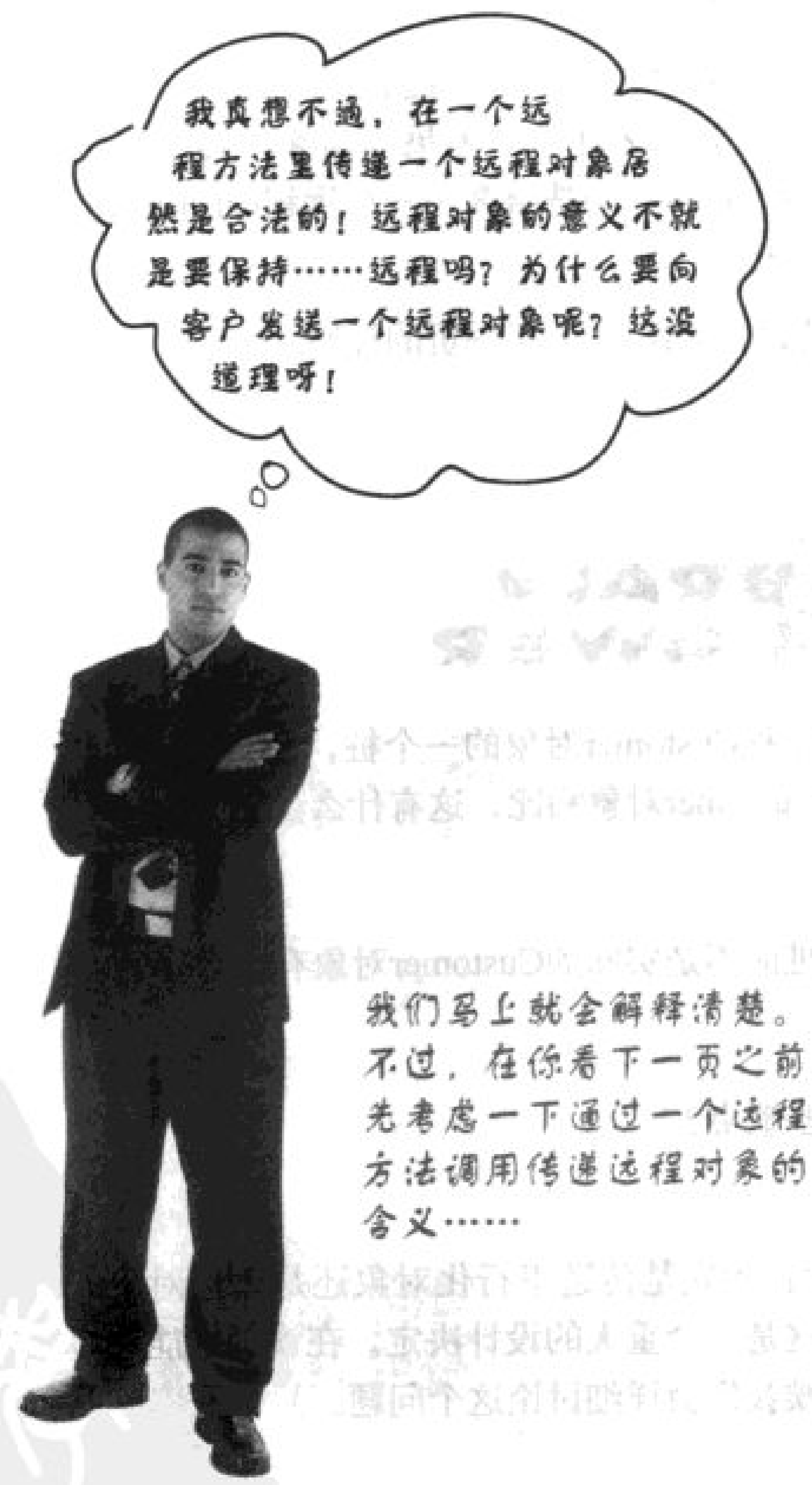
**问：** **我的类必须显式地实现Serializable吗？能不能从超类继承这种可串行性呢？**

**答：** 要记住Java里有一个IS-A规则：你的父类（超类）是什么，你就是什么。如果Dog扩展了Animal，而Animal实现了Serializable，那么无论Dog是否显式声明实现Serializable，Dog都是Serializable。

不过，即使超类是Serializable，仍然可以显式地把你的类声明为Serializable，一般认为这是一种好的实践做法，这样其他人在看你的类API时就不用在类继承树里到处搜寻，看这个类上面是否有哪个超类是Serializable。

**要记住，远程方法的参数和返回值必须是以下某种类型：**

- 基本类型
- Serializable 对象
- 基本类型或Serializable对象组成的一个数组或集合
- Remote（远程）对象



## 通过远程方法调用传递一个远程对象

在远程方法调用中传送一个远程对象是没道理的。毕竟，远程对象的意义就是保持远程。要由“住”在……别的地方的客户来访问，也就是说，访问它的客户应该在另一个堆中。

但是，如果你想向一个远程客户传递另一个远程对象的引用，会怎么样呢？如果不向客户传递一个完整的Customer副本，而是给客户一个远程Customer的桩，又会怎么样？

先想想看，然后再看下一页的说明。

**通过一个远程方法传递远程对象时，Java实际上发送的是这个远程对象的桩。**

**换句话说，在运行时，远程对象还待在它原来的地方，在网络上传送的只是它的桩而已。**

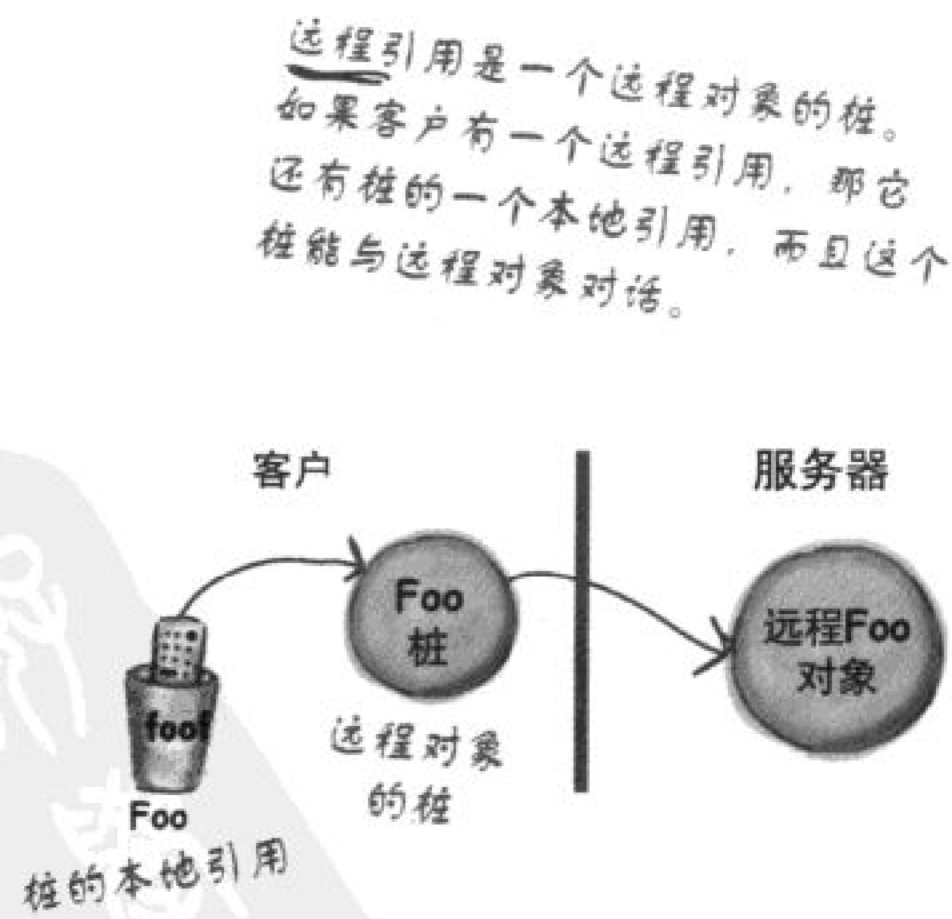
**BRAIN POWER**

传递远程Customer对象的一个桩，与传递一个非远程Customer对象相比，这有什么含义？

传递桩而不是实际的Customer对象有什么好处？

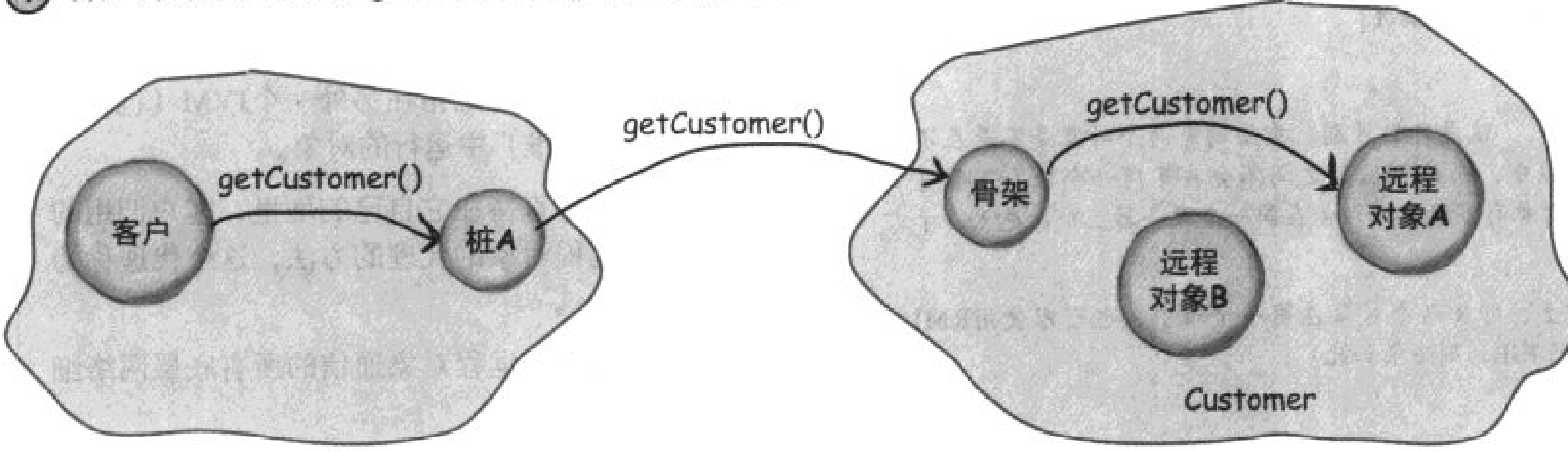
有哪些缺点？

（注意：到底是传递串行化对象还是远程对象的桩，这是一个重大的设计决定。在谈到性能和模式时候我们会详细讨论这个问题。）



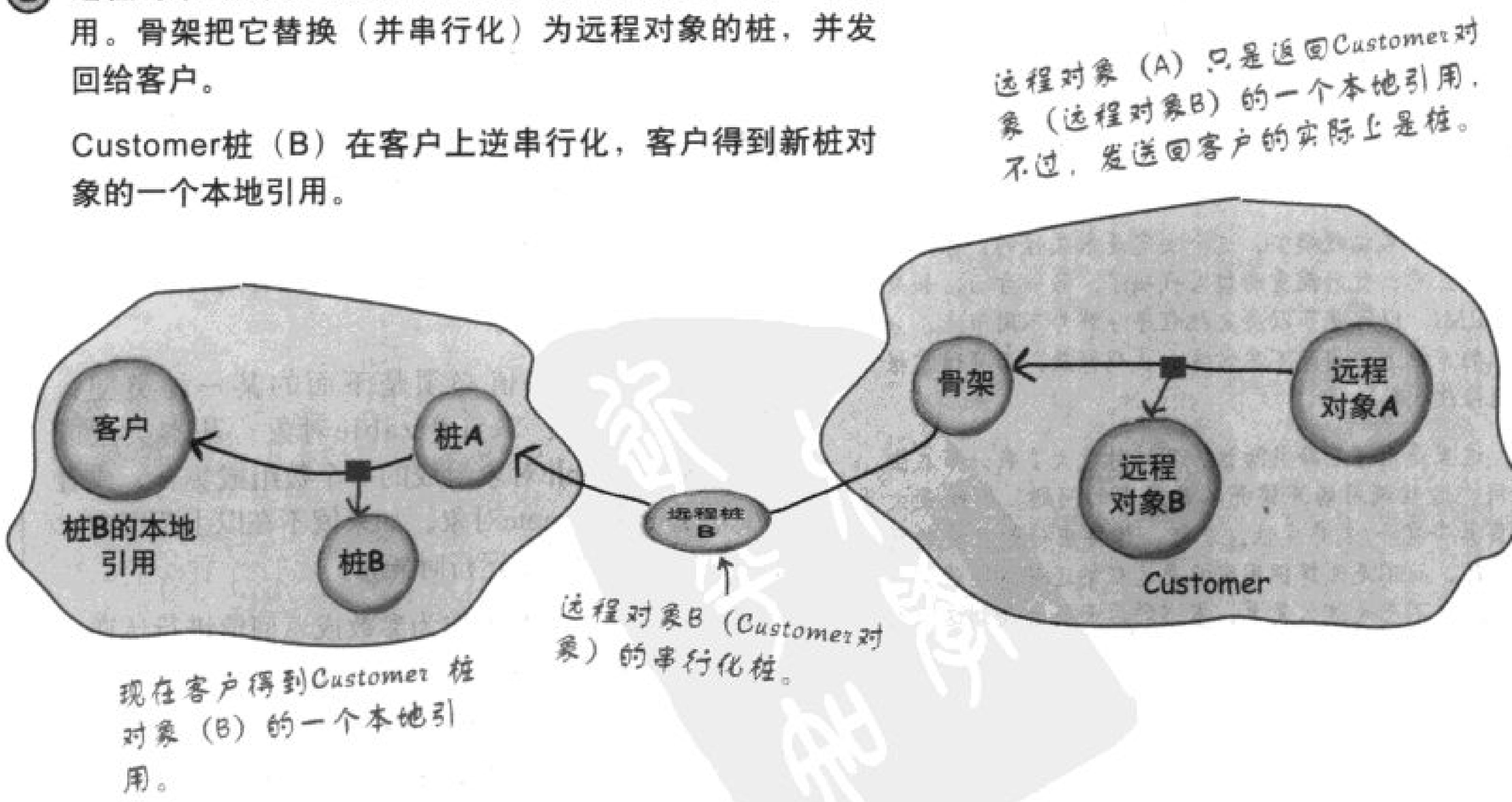
# 返回值是远程对象时……

① 客户调用远程对象的getCustomer()（使用桩A）。




② 远程对象A返回Customer对象（远程对象B）的一个引用。骨架把它替换（并串行化）为远程对象的桩，并发回给客户。

Customer桩（B）在客户上逆串行化，客户得到新桩对象的一个本地引用。



## there are no Dumb Questions

**问：** **如果客户对象和远程对象在不同的JVM中运行，但是所在的物理主机是同一个，会怎么样呢？换句话说，它们都在同一个服务器上的（不同）Java程序中运行时，会出现什么情况？**

**答：** 这没什么区别。重要的是两个对象是不是在不同的堆中，而JVM彼此之间不会共享同一个堆，不管这两个对象有多亲密（即使在同一个服务器上），也无济于事。

实际上，即使两个对象在同一个堆中，也可以使用RMI（对于EJB，则必须如此）。

**问：** **如果不需要使用RMI，那你为什么还要使用它呢？这样一来，不就会带来远程调用的开销吗？**

**答：** 这一点在后面会做更详细的解释，不过可以指出，主要原因是因为，如果方法调用没有使用RMI，你就会把自己的设计"封锁"起来，将无法把对象分布到网络中的不同位置（甚至不能分布在同一个服务器上的不同地方）。换句话说，如果没有RMI，方法调用涉及的对象就必须在同一个堆中。

对于一个分布式编程模型，这个决定是永久性的，你以后要想改变主意，就只能重新编写代码了。另一方面，如果使用了RMI，以后就可以决定把程序分解为不同节点，分布在你的系统上，为此不用做多少代码修改，而且通常根本不用修改代码。

所以，这里是用性能换灵活性，不过对于大多数分布式企业应用，远程调用的开销并不是最大的问题。影响最大的还要算带宽和/或并发性。总之，你要应付的更大的问题是性能，而不是选择调用究竟是远程的还是本地的。不过，事情并不总是这么简单，本书的后面还会谈到这个问题。

### 要点

- EJB使用了Java RMI（Remote Method Invocation，远程方法），从而使远程客户可以访问你的bean。
- 这里的远程客户是指在另外一个JVM（也称为一个不同的堆）中运行的对象。
- 远程对象一直待在它自己的堆里，客户调用的是这个远程对象的代理的方法，这个代理称为桩（stub）。
- 桩对象处理与远程对象通信的所有底层网络细节。
- 客户想调用一个远程对象上的方法时，实际上会在桩上调用这个方法。桩与客户在同一个堆中。
- 对客户来说，远程方法调用和本地方法调用是一样的，只不过远程方法可能抛出一个RemoteException异常（这是一个受查异常）。
- 桩把方法参数包装起来，并把调用的有关信息发送给服务器上的一个骨架。骨架对象本身是可选的，不过服务器上必须有"人"来做骨架的工作。我们不关心是谁（或者是什么）具体做骨架的工作。
- 参数和返回值必须是下面的某一种类型：基本类型、Serializable对象、基本类型或Serializable对象组成的一个数组或集合，或者是一个Remote对象。如果值不在以上范围内，就会出现一个运行时异常。
- 如果把一个对象作为参数或返回值进行传递，这个对象会按串行化副本发送，然后在远程对象的本地堆上逆串行化。
- 如果把一个远程对象作为参数或返回值进行传递，实际传递的是这个对象的桩。


哦，我的天哪！差点忘了远程方法调用最重要的一件事！把一个串行化对象作为参数或返回值传递时，最好确保对方有所传递类型的类文件。如果那里没有所需的类，对象就无法得到逆串行化，你就该犯愁了！
需要的类里还包括桩类。如果客户没有桩对象的相应类，要想达到目的也没戏。

# 远程对象和桩必须有哪些共同之处?

客户怎么知道要调用哪些方法?

桩怎么知道远程对象有哪些方法?

要记住，如果桩想假装成远程对象，桩就必须有远程对象同样的方法。

你肯定知道这个问题的答案。

这就是利用一个接口。

这样一来，分布式环境中的所有方法都要展现给客户。

我们称它为业务接口，因为其中有客户想调用的业务方法。从技术上讲，远程对象的业务接口必须是一个远程接口，这有点奇怪。

**都是先从一个远程接口开始。远程对象和桩都实现同样的接口……这个接口包括了客户想调用的方法。**

**远程接口必须扩展java.rmi.Remote，而且每个方法都必须声明一个RemoteException。**

要作为一个远程接口，必须满足以下3条原则：

- **必须扩展**java.rmi.Remote
- **每个方法都必须声明一个**java.rmi.RemoteException
- **参数和返回类型必须是可传递的（必须是**Serializable**对象、基本类型等等）**

```
import java.rmi.*;

public interface DiceRoller extends Remote {

    public int rollDice() throws RemoteException;
}
```

RemoteException和Remote接口都在java.rmi包中

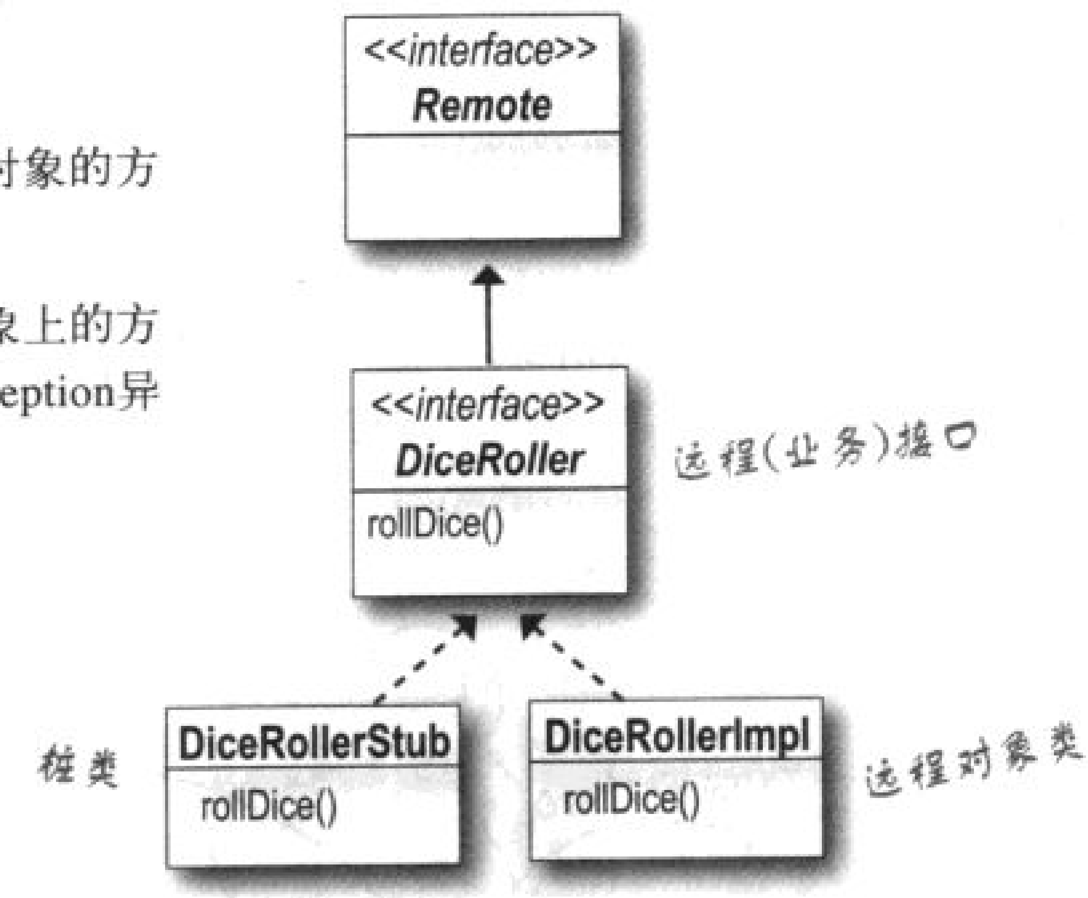
远程接口必须扩展java.rmi.Remote（这个接口里没有任何方法）。

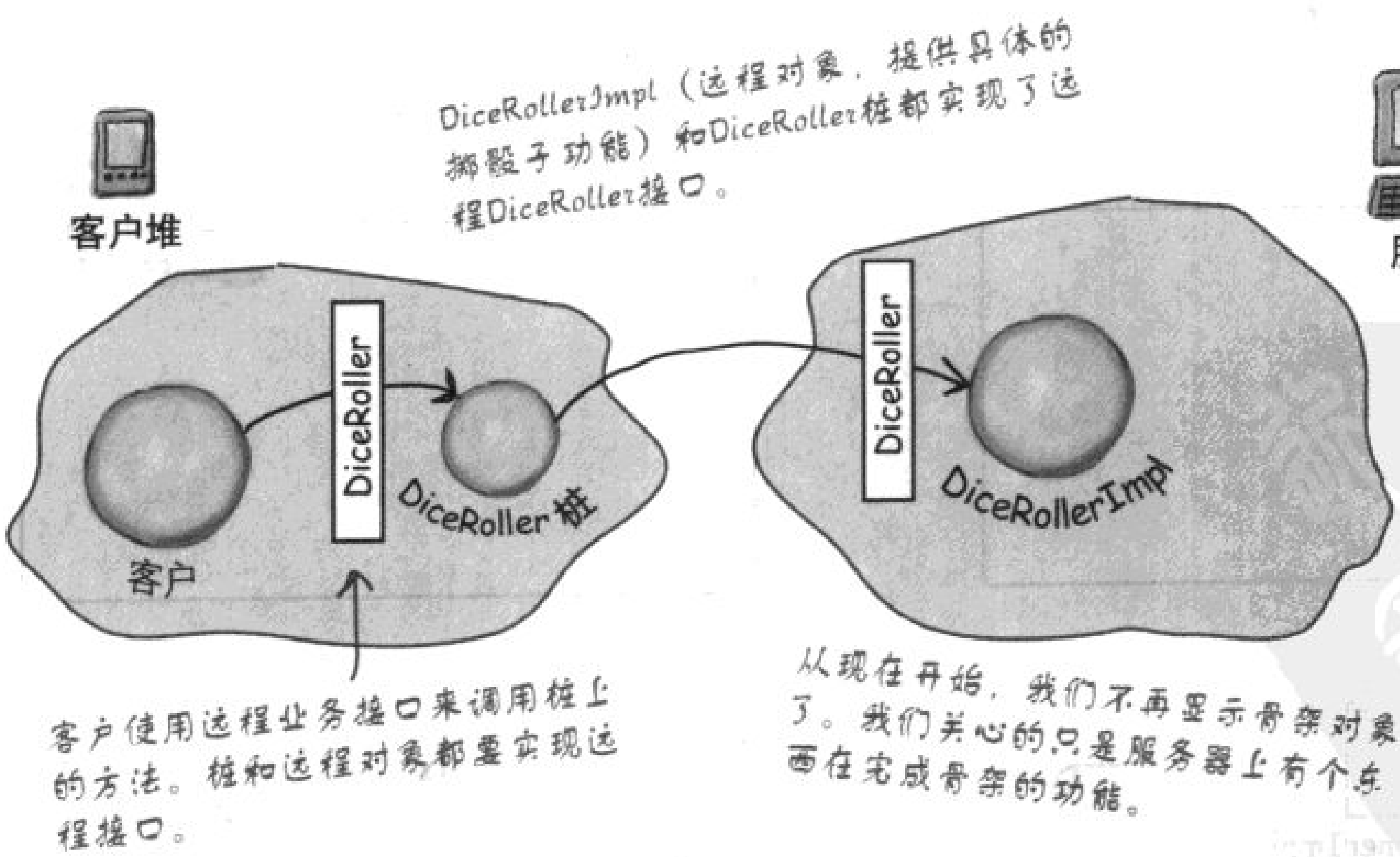
你的所有方法都必须声明一个RemoteException。

# 客户通过远程业务接口调用桩上的业务方法

要记住，对客户来说，他觉得他调用的就是真正远程对象的方法。这个对象有他想调用的方法。

不过实际上并不是这样，客户并没有直接调用远程对象上的方法，惟一能提醒他这一点的是，他必须处理RemoteException异常。

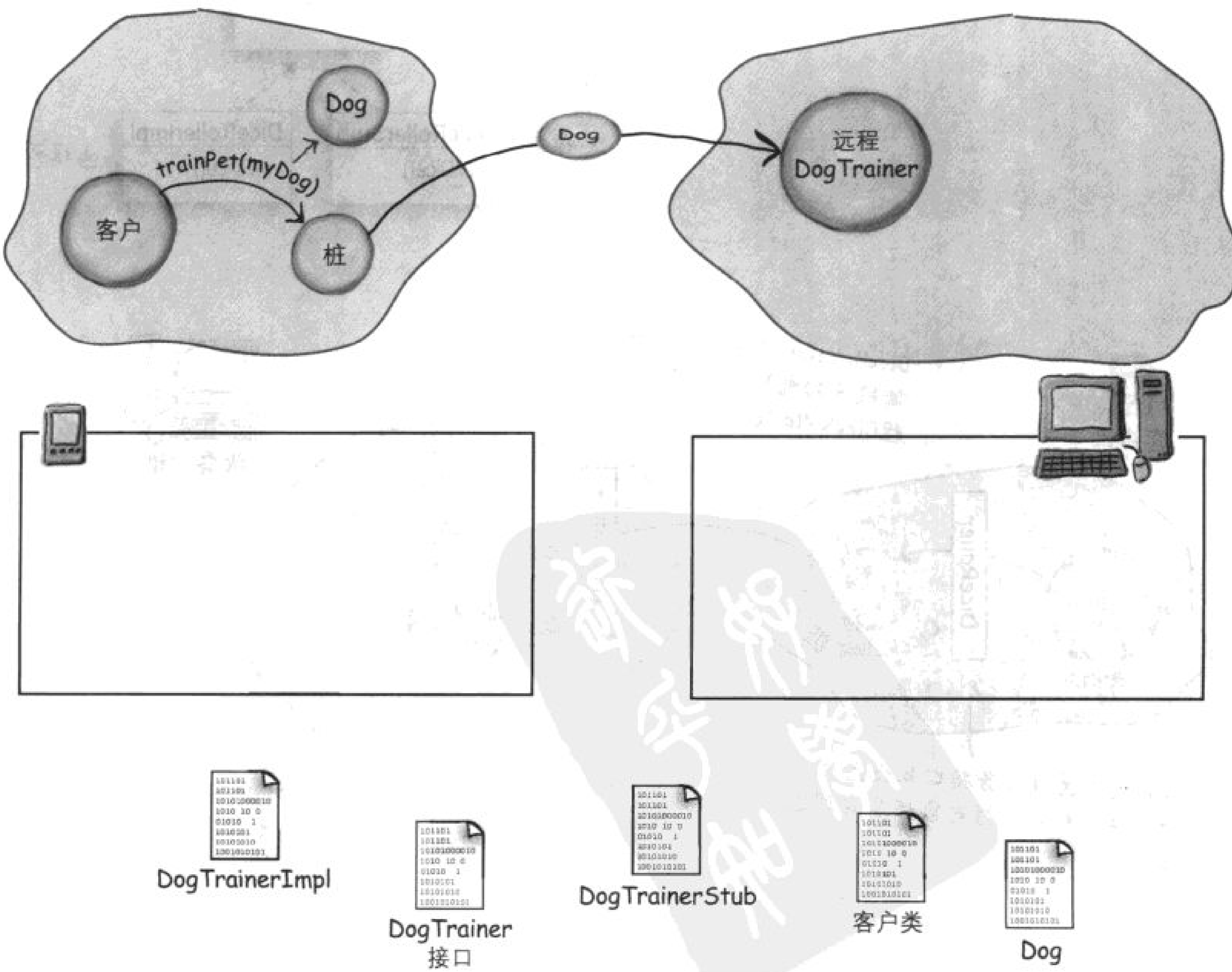


**Sharpen your pencil**

这里的练习涉及到EJB开发人员最常犯的一个错误。所以不要跳过去不做！

针对以下情况，把下面的类画到适当的框里，这些类可能必须在客户上，或者必须在服务器上，还有可能在客户和服务器上都有（可以重用类）。这个图有所简化，所以这里显示得并不全，有些参与的对象没有画出来。

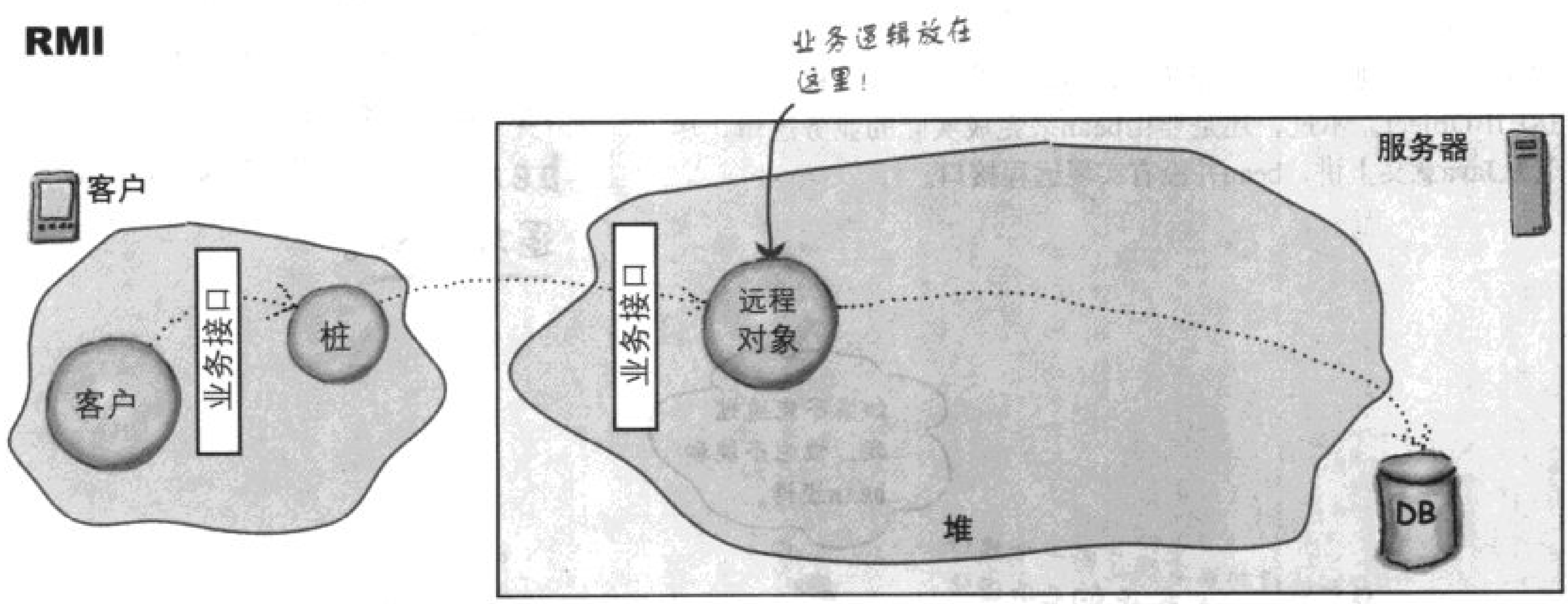


# EJB如何使用RMI

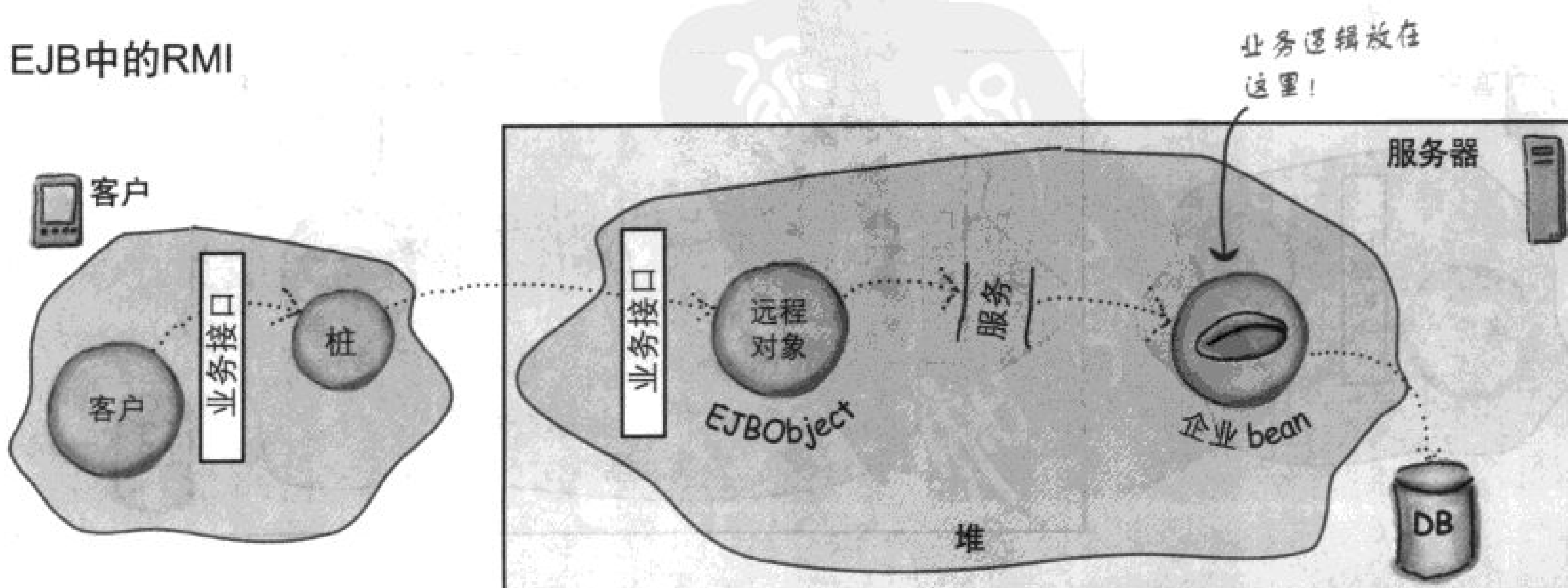
在EJB中，客户到企业应用的入口点几乎都是通过一个远程对象的引用（桩）。不错，可能也会使用一个本地客户，而且有时这还是必需的。所谓本地客户是指，客户与bean在同一个堆中而且不使用RMI来调用业务方法，但是这种情况实在是少之又少。

所以大多数客户到bean的通信中，RMI都是核心。不过，从第1章中的提示可以看到，EJB体系结构比简单的客户到桩再到远程对象的情况要复杂一些。在EJB中，bean（客户就是想在这个bean上面调用业务方法）不是远程的（译者注：就是说bean没有真正实现远程接口）！

**RMI**




EJB中的RMI



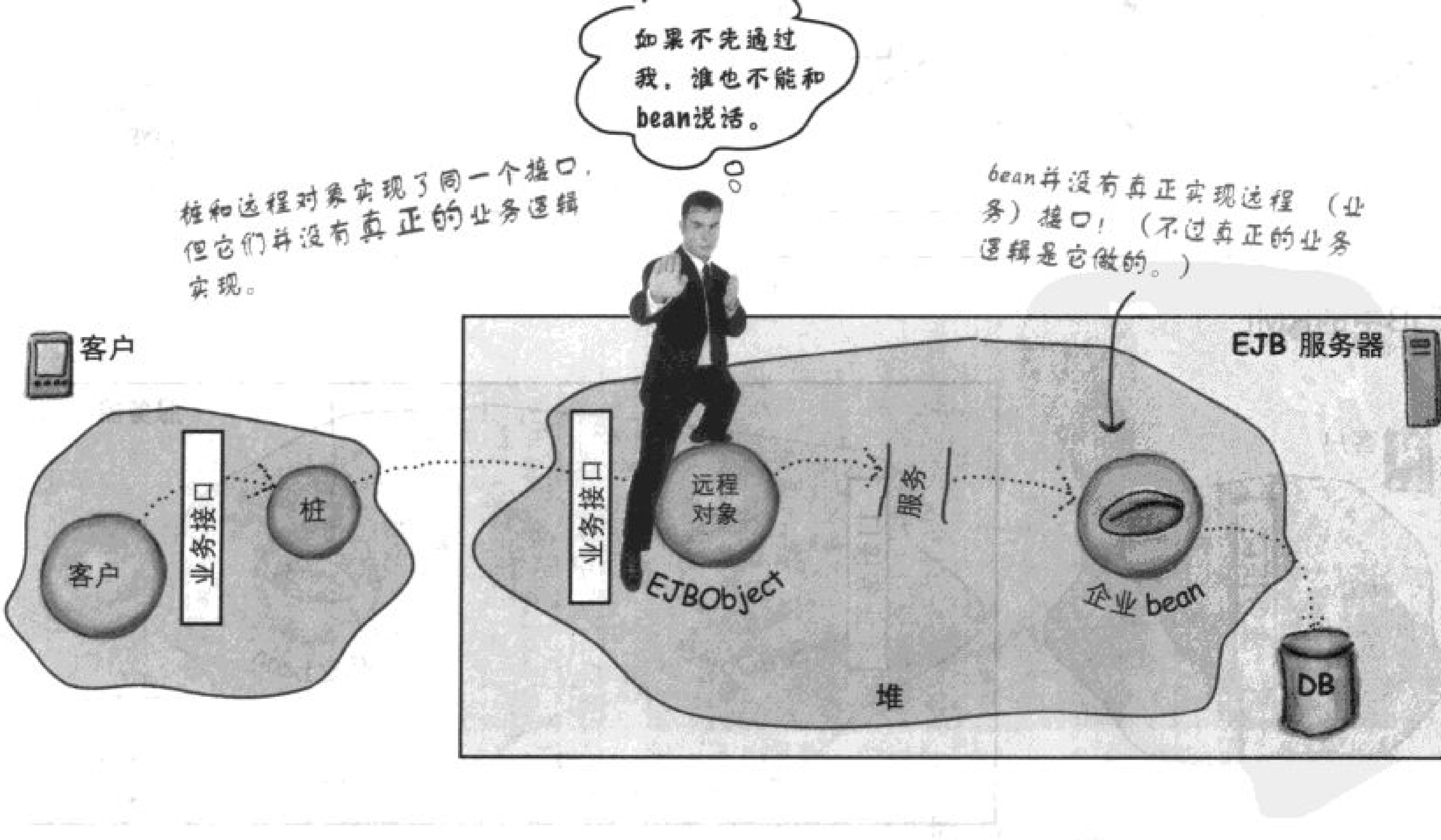
# 远程对象不是bean，它是bean的保镖——EJBObject

要记住，在EJB中，远程对象（EJBObject）是bean的保镖。bean只是坐在一边，不受客户调用的任何干扰，而EJBObject会实现远程接口，并得到远程调用。一旦调用到达EJBObject，服务器就会跳出来介入，插入所有服务，比如安全（这个客户得到授权了吗，可以调用这个方法吗？）、事务（这个调用是一个现有事务的一部分吗，或者是不是应该开始另一个事务？）以及持久存储（运行客户的方法之前，这个bean需不需要从数据库加载信息？）。

EJBObject实现了远程业务接口，所以客户发出的远程调用会到达EJBObject。不过，还是要由bean来完成实际的业务逻辑，尽管从Java意义上讲，bean并没有实现远程接口。

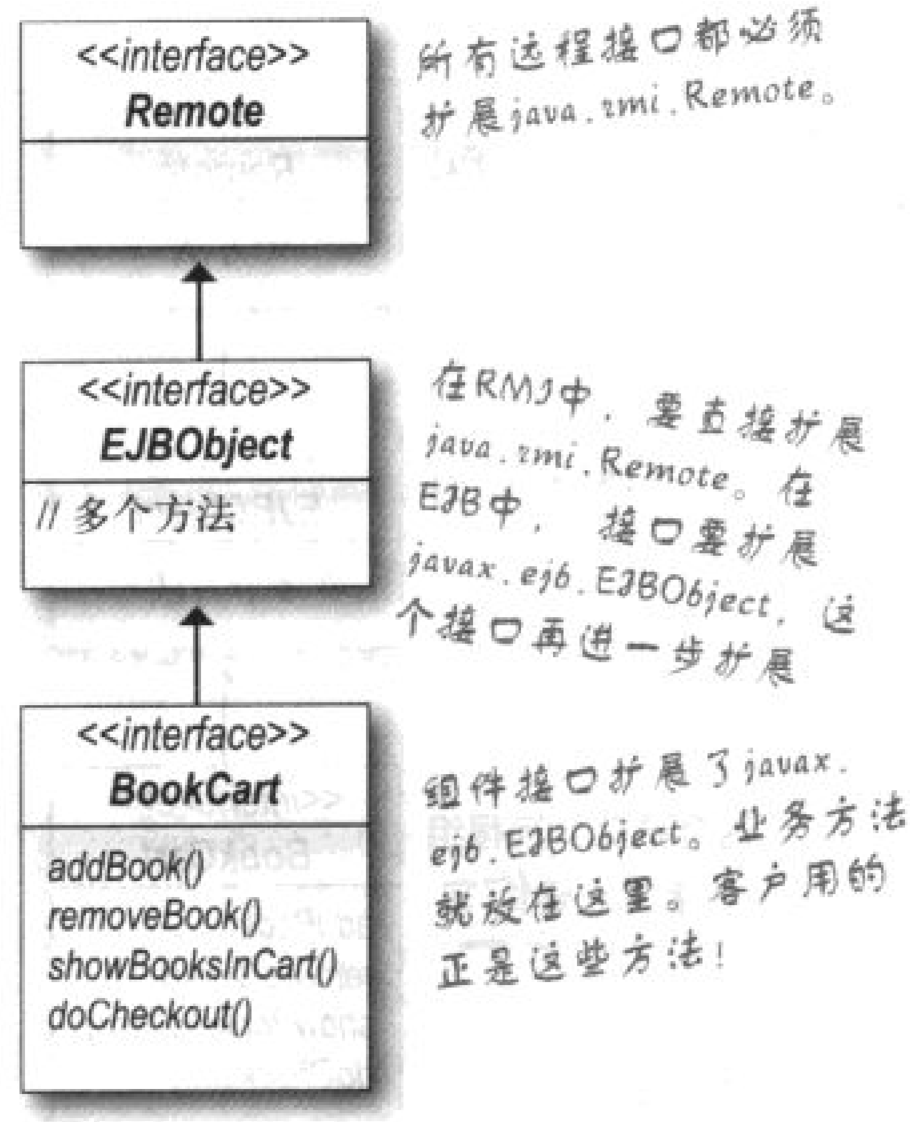
**远程对象和桩都实现了同一个接口，即业务接口（也称为组件接口），不过没有实际的业务逻辑行为。**

**bean类没有实现业务接口（从正式的Java语法角度来看），不过bean才有真正的业务逻辑功能。**



# 组件接口

在EJB中，业务接口称为组件接口。你要在这里把业务方法展现给客户。RMI接口和远程组件接口之间的主要差别在于，在EJB中，远程组件接口扩展的是javax.ejb.EJBObject而不是java.rmi.Remote。




**要点：**

1. 如果一个接口的继承树里有java.rmi.Remote，这个接口就是一个远程接口。

2. EJBObject接口扩展了Remote，所以EJBObject是一个远程接口。

3. 你的远程组件接口必须扩展EJBObject接口。

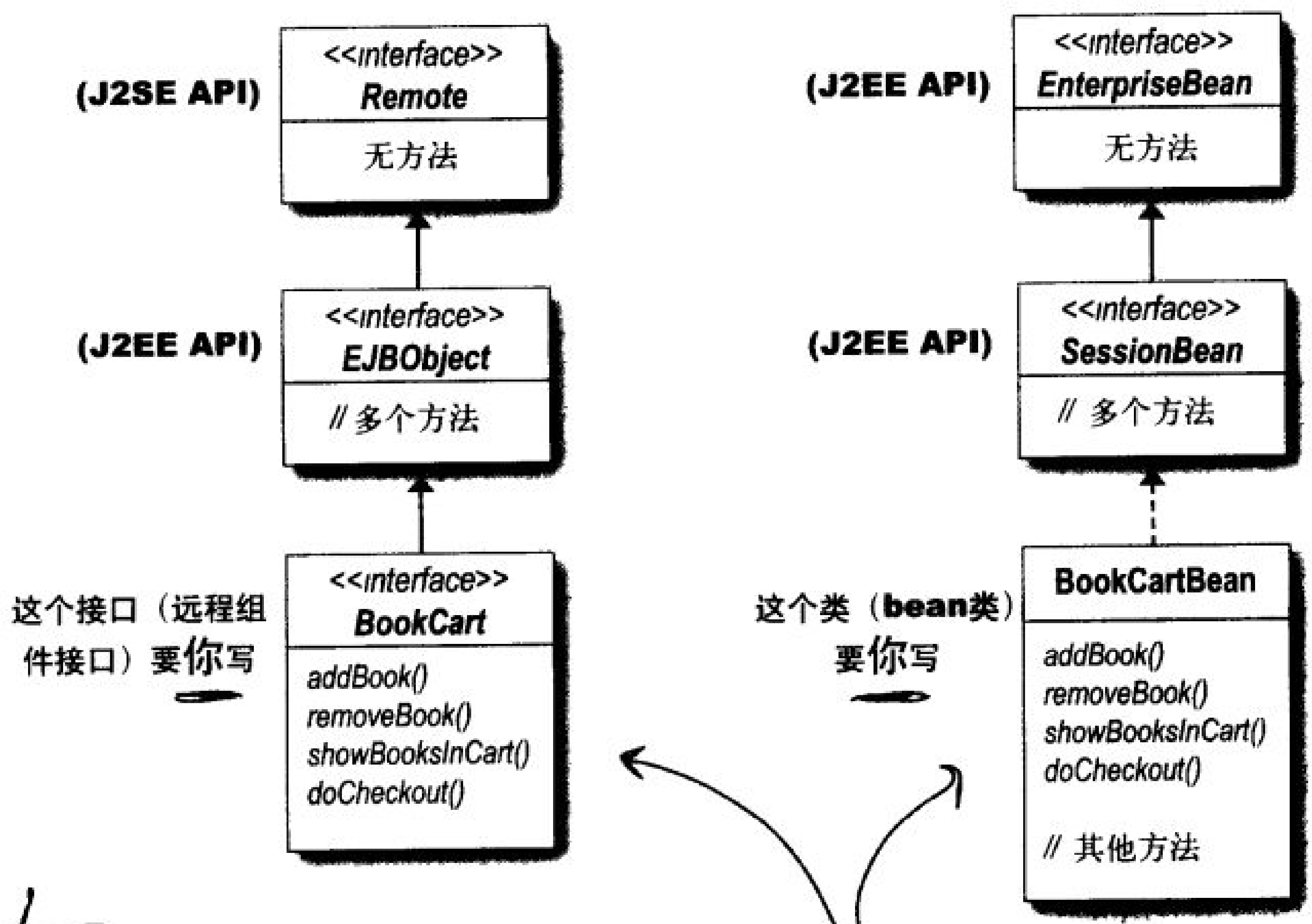
   （你也可以有一个本地组件接口，本地组件接口的规则与此完全不同，不过这些内容到客户视图一章再做介绍。）

4. 通过组件接口把业务方法展现给客户。

5. EJBObject接口增加另外一些方法以供客户使用（后面会看到）。

**不管是谁实现BookCart接口，都不光要实现BookCart的所有方法，还要实现EJBObject的全部方法。EJBObject接口增加了所有EJB客户都可能需要的一些方法。**

# bean类的地位




同样的方法，但是没有直接的Java关联。BookCartBean没有实现BookCart接口！（不过看上去好像实现了一样……）

bean开发人员必须确保bean类中的方法与组件接口中的方法相匹配，就好像bean类实现了组件接口一样。

**你必须了解这些接口！**

考试要求你很清楚每个方法在哪个接口里。例如，你要知道哪些方法在EJBObject里，哪些方法在SessionBean里。而且你必须知道这些方法的准确签名。在后面的几章里我们会深入分析这些接口，所以在读到这些内容时，你一定要特别注意！

## there are no Dumb Questions

**问：不知道我这样理解对不对……接口还可以扩展其他接口吗？**

**答：** 对、接口也有自己的继承树。实际上，甚至类做不到的一些事情，接口都能做到，类不能多重继承，而接口可以扩展多个接口！

```
interface Cart extends EJBObject,CartBusiness
```

**问：但是，一个接口扩展另一个接口到底是什么意思？扩展就意味着继承，不过什么是接口继承呢？**

**答：** 一个接口扩展另一个接口时，它会继承后者的所有一切。不管是谁实现了一个接口，它就不光要实现这个接口的方法，还要实现该接口所扩展的每一个接口的方法……而且要实现这个接口的继承树中每一个超接口的所有方法。

所以，在这个例子中，只要实现了BookCart，就必须还要实现EJBObject的方法。

实现一个接口时，必须实现这个接口从它的超接口继承来的所有方法。

所以，不管是谁实现了BookCart接口，都必须实现BookCart的方法及EJBObject的方法。

**问：为什么bean不实现远程/业务接口呢？接口实现的意义不就在于用编译器来保证正确，支持Java类型检查吗？**

**答：** 这个问题你以前已经问过了。不过，谁都会忘事，所以我们再给你解释一遍。bean没有实现远程接口，原因在于bean绝对不会作为一个远程对象（从Java RMI的角度讲）。换句话说，你绝对不希望有人拥有实际bean的桩！如果你真的不小心让一个远程引用（也就是桩）溜到外面去了，就会让EJB功亏一篑，也就是说，这会违反EJB的主旨！如果让客户直接与bean对话，那服务器就没办法应用它的服务了，如果你真的不需要这些服务……我们好像扯远了。

从技术上讲，bean实现远程接口是合法的，不过这确实是一个不好的想法，因为这样一来，你可能会犯一些在编译时未能捕获到的错误（但是这些问题会在以后爆发）。而且你也不需要这样做，因为几乎所有bean开发工具（几乎每一个支持bean的IDE）都了解bean、EJBObject和远程接口之间的关系，这些工具可以保证组件接口和bean类有匹配的方法。

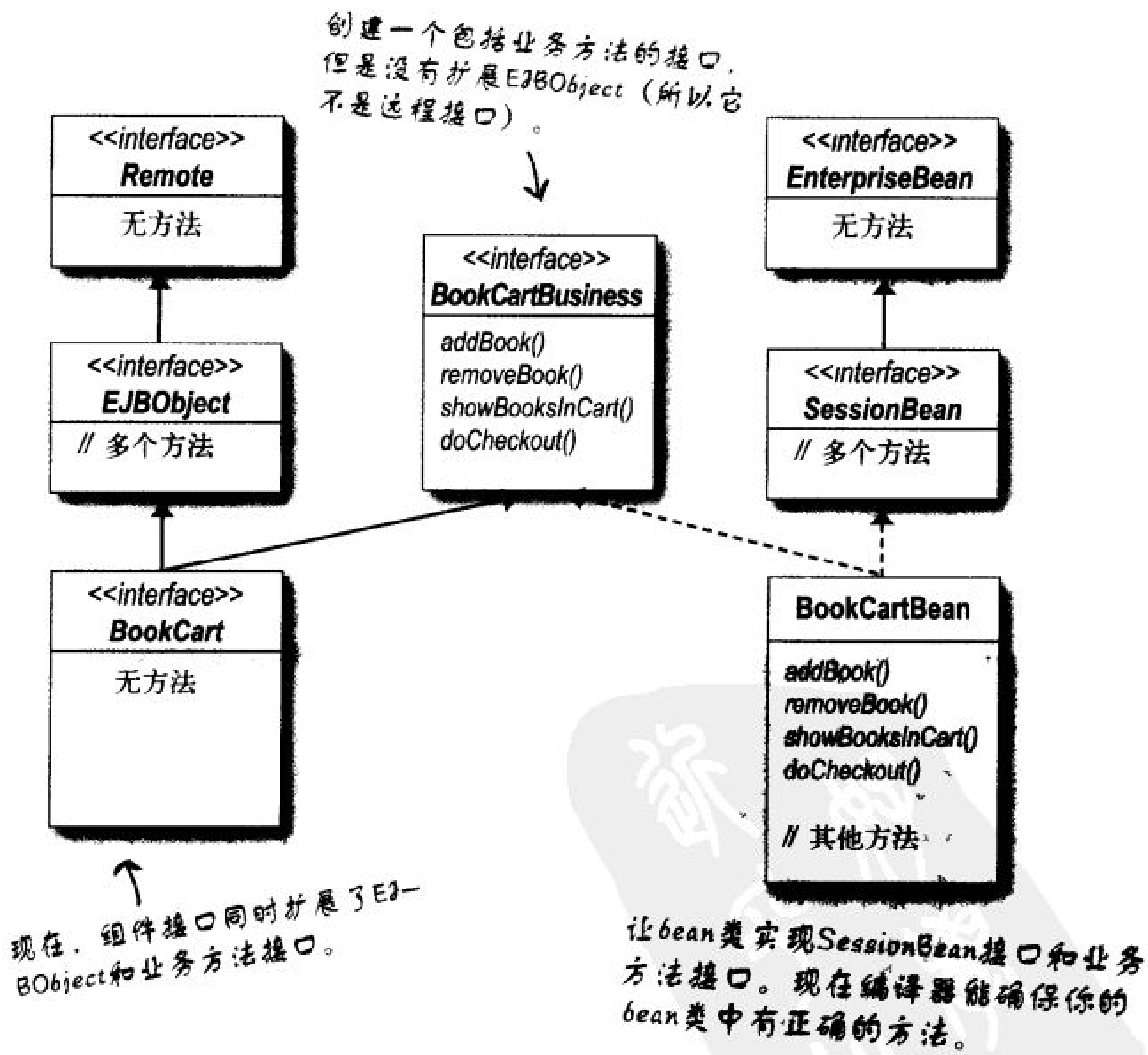
**问：但是，如果我还是觉得这样不太好，这种想法和我原来的Java思路背道而驰，也和我编写代码的原则不符。肯定有我能做的事情吧？**

**答：** 相信我，“你肯定会使用工具，所以真的不用你自己做什么，真的”，不过如果你一再坚持，也确实可以做点事情，这样也许会让你感觉好点。下一页会谈到你能做什么，不过可能你自己也能想出来。

但是，无论如何，我们还是坚持我们的意见，大多数开发人员都不需要这样做。




## 这个设计是给那些对bean没有实现业务方法接口感觉不舒服的人准备的……






## 谁真正实现组件接口的类呢？换句话说，谁来创建EJBObject类？

答案是由容器来建。你来声明方法，但是由容器实现你的组件接口。要记住，你的组件接口是一个扩展了EJBObject的接口，所以容器不光要实现你的业务方法，还要实现EJBObject的方法（到目前为止还没有看到过）。

**问：** 不过，容器怎么知道这些方法里放什么呢？这些方法是我声明的，又不是容器声明的……

**答：** 记住，容器不实现实际的业务逻辑。这些业务方法的真正功能由你的bean类来实现，而bean类正是要你实现的类。实现组件接口的类会作为EJBObject，这是保镖，也是远程对象。要知道，EJBObject只是假装是个bean。它会对来自客户的远程方法调用做出响应（通过桩），而EJBObject唯一的任务就是捕获客户对bean的调用。此后，就要由容器/服务器来接管了。

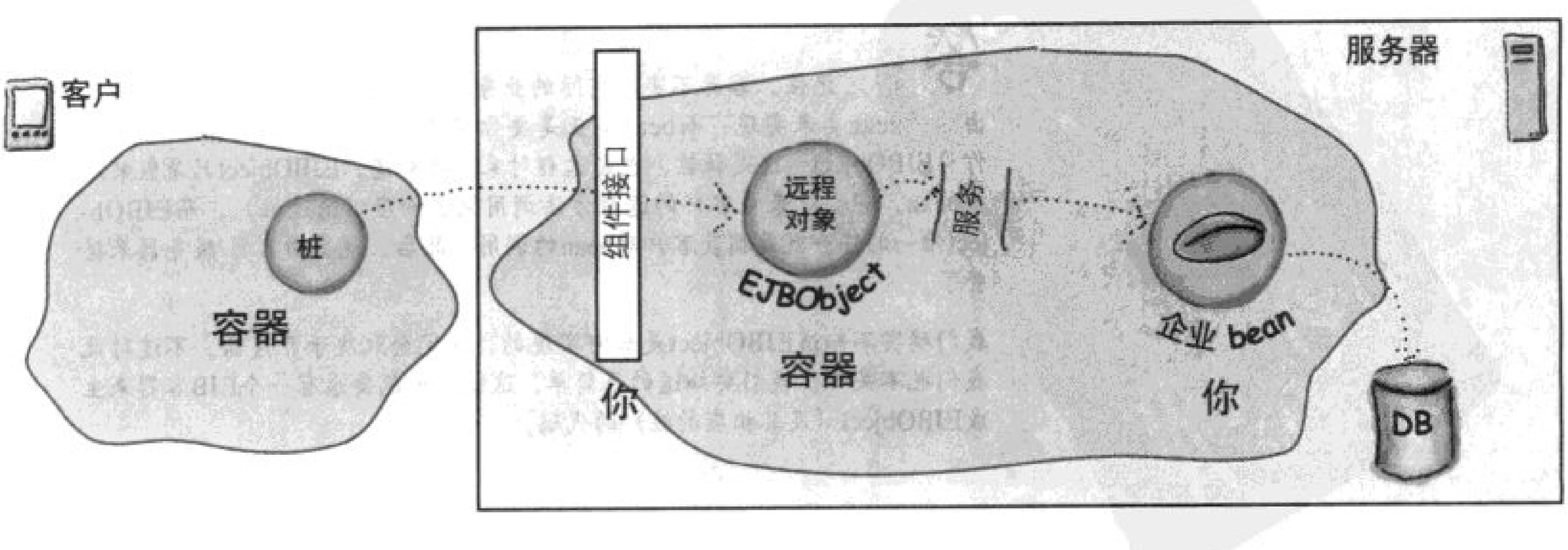
我们确实不知道EJBObject是如何实现的，这完全取决于开发商。不过对此我们也不关心。我们要知道的很简单，这就是规范要求有一个EJB容器来生成EJBObject（及其相应的桩）的代码。

# 谁创建什么？

对于一个有远程客户视图的bean（也就是说，这个bean可以由远程客户访问），必须编写组件接口和bean类，这是你知道的。不过，必须有人编写一个实现组件接口的类（来建立远程EJBObject),而且要有人建立与EJBObject对应的桩。这个人就是容器。而且，尽管我们还没有谈到Home，但为了完整起见，下面也列出了Home的相关部分。

| 你 | 容器 |
|---|---|
| ① 组件接口<br>（扩展javax.ejb.EJBObject） | ① EJBObject类<br>（实现你的组件接口） |
| ② bean类<br>（实现javax.ejb.SessionBean或javax.ejb.EntityBean） | ② EJBObject桩类<br>（实现你的组件接口，并知道怎么与EJBObject对话） |
| ③ Home接口<br>（扩展javax.ejb.EJBHome；下一页会讨论这个内容） | ③ Home类<br>（实现你的Home接口） |
| | ④ Home桩类<br>（实现你的Home接口，并知道怎么与Home对话） |

我们还没有讨论Home，所以这个图里没有显示Home。





**EJBObject**: 嘿，小bean……每次我都挡在你前面，你不烦吗？你不想直接和某个人谈话吗？

**Bean**: 不，我多重要啊，而且我太宝贵了。我肯定不会亲自接收对我的调用。这是你们这些人做的事情。

**EJBObject**: 你们这些人？

**Bean**: 对，你们这些为容器效力的人。你、Home、桩，所有这些人。我的任务是处理复杂的业务逻辑。这些功能至关重要，它们意味着一个企业环境的成败。

**EJBObject**: （嘘声）。就算这样，你是有一些重要的方法，但是我还是看不出为什么每个调用都需要我。

**Bean**: 想想看，我的工作那么重要，所以不能被那些根本没有业务调用的客户打断。你果真认为我要检查每个调用者的安全权限吗？我难道不该做些更好的事情吗？

**EJBObject**: 好吧，这确实只是一个安全问题，不过我不明白为什么你不能有这样一些代码来检查调用者的安全访问呢？这样可以节省很多开销（比如说，可以节省我的开销）。

**Bean**: 首先，让你来接收我的调用，原因有很多，安全性只是其中的一个原因。稍后我就会解释其他原因。不过，至于放些代码来完成我自己的检查，这个事情我当然可以做，如果程序员希望我这样做，也没什么不行。不过，这往往不是处理安全性的办法。

**EJBObject**: 在你自己的代码里处理安全检查有什么不对吗？

**Bean**: 你真的不知道吗？[转转眼珠]首先，如果我这里放上安全检查代码，我就无法重用了。EJB的意义就在于可以在运行时配置和定制bean，而无需重写代码。如果把安全写到我的Java代码里了，那么除非改动代码，否则无法对安全做任何调整。谁希望这样呢？

**EJBObject**: 我想这有道理。你把安全检查放到XML部署描述文件里，然后我拿到调用，服务器可以查看客户是否得到了适当的授权。但是，如果你不存在安全问题呢？如果部署你的人不在乎到底谁调用方法，又会怎么样呢？

**Bean**: 你真是个愚木疙瘩。不过至少你能拉动那么重的对象，暂且原谅你吧……你再想想其他一些重要的事情，比如事务和持久存储。

**EJBObject**: 哦，我把事务忘了。对，这也有道理。服务器调用你的方法之前必须明确是否已经有一个事务上下文，这样你的方法就能在一个事务中运行了。可以是你自己的事务，也可能是调用者的事务……

**Bean**: 是啊。

**EJBObject**: 但是哪有持久存储的事呀？

**Bean**: 嗯，先想想实体bean。如果我是一个实体bean，这说明我表示底层持久存储库中的某个实体……

**EJBObject**: 先等等，你说持久存储库？是不是就是指数据库？为什么不直接说数据库呢？

**Bean**: 这还不知道，“持久存储库”这个词可不是“数据库”的同义词。数据库只是持久存储库的一个例子。不过，如果你觉得把持久存储库想成数据库更容易接受，现在暂且这样想吧。但是，我说过，如果我表示一个实体，比如，一个名字叫Tom Duff的顾客，客户调用我的getAddress()方法时会发生什么呢？服务器不会把这个调用传给我！

**EJBObject**: 为什么呢？

**Bean**: 因为我必须先从数据库把Tom Duff的信息加载进来！

**EJBObject**: 为什么呢？

**Bean**: 因为我可能会返回一个不合法的地址！除非我还和这个客户在以前的事务中，在这种情况下，服务器告诉我运行getAddress()方法之前，必须先告诉我要加载数据库里Tom Duff的信息。否则，如果不是这种情况，谁知道我会返回什么呢？好了，就到这里吧，以后有空再谈。

## there are no Dumb Questions

**问：** 容器怎么创建EJBObject、Home还有桩，什么时候创建？

**答：** 部署一个bean时，容器会查看部署描述文件，根据它得到bean。要记住，部署描述文件给出了远程组件接口（EJBObject）和远程home接口的完全限定名。所以，一旦容器得到这些接口，它就会生成代码，建立实现这两个接口的两个类。因为它们都是远程的，所以容器还会创建客户桩，这些客户桩知道怎么反过来与远程对象通信。

**问：** 这些桩是普通的RMI桩吗？我发现使用RI部署时，它会打印一条消息说正在运行rmic……

**答：** 容器会尽其所能来创建桩；惟一的要求是桩必须是RMI-IIOP兼容的。实际上，服务器可以使用一种称为动态代理的对象来实现桩功能，不过到底怎么实现我们并不关心。在谈到"桩"的时候，我们是指有桩行为的任何对象。它到底是一个RMI桩还是其他的什么东西，这只是服务器的一个实现细节。在下一章中我们会更详细地介绍这个内容。关键是：你确实不知道桩类代码是什么样子。出于这个原因，你也不知道EJBObject和Home是怎么实现的。你也不需要知道。

可能你有一个EJB容器允许你查看（甚至修改）它生成的源代码，不过别指望。如果我是你，我们就不会这样做，就算可以做也不做。

**问：** 那么，这些类都要由开发商的实现决定啰？

**答：** 对！开发商在实现这些类时有各种各样的选择，它可以使用桩和/或Home以及EJBObject来提高性能。不过，重申一句，这不需要你掺和，甚至无需你知道。

规范中惟一要保证的需求（而且必须遵守）是：远程对象必须遵循RMI-IIOP的规则，这说明Java的远程方法调用用的是IIOP（CORBA标准）通信协议。

**问：** 既然提到了，那么RMI-IIOP与常规的RMI有什么区别呢？

**答：** 普通的RMI使用JRMP作为通信协议。而IIOP允许远程对象通过CORBA实现互操作（这本书里不会对CORBA谈太多；它超出了这本书的范畴，而且考试肯定不会考，不过会在书中偶尔见到一些有关CORBA的小问题）。

这样你的对象就有机会得到客户的访问，甚至是非Java客户。IIOP指定了事务和安全信息怎么随方法调用传播，你的容器可能会充分利用这一点。

大多数情况下，你很少会注意到普通RMI和RMI-IIOP之间的差别。不过……确实有些地方是有区别的，其中一个区别在考试中肯定会考，这就是需要窄化桩。有关窄化的内容将在有关客户视图的下一章详细介绍。对现在来说，只要知道这是客户必须对EJB桩做的事情，而对普通的Java桩无需这样做，这是因为EJB规范指出，要假设桩使用的是IIOP，所以可能是一种不同的桩（因此要窄化）。

**问：** 什么时候才谈到Home？

**答：** 下一页。

**问：** 为什么这么晚才讲到Home？难道Home不重要吗？

**答：** Home非常重要，要得到实现组件接口的类的引用，Home这往往是惟一的途径。换句话说，要使用Home来得到一个EJBObject桩（为远程客户得到一个EJBObject桩，这是我们前面一直谈论的）。

对于实体bean，Home还可以有一个更重要的角色，这一点我们后面就会看到，不过即使是实体bean，Home的主要用途还是获得EJBObject桩。得到了EJBObject桩之后，客户和bean之间的大部分通信都是通过EJBObject而不是Home。实际上，大多数情况下，客户使用Home只是为了得到EJBObject引用，然后Home引用就会被丢掉，不再需要了。

# bean Home

每个会话bean和实体bean都有一个Home。

消息驱动bean没有home，因为消息驱动bean没有客户视图（换句话说，客户是无法得到一个消息驱动bean的引用的）。

Home有一个主要任务：就是向外分发bean组件接口的引用。对于一个会话bean，bean的Home只是干这个。不过，对于实体bean，Home的作用就要大多了。

已经部署的每个bean都有自己的Home，这个Home要对该类型的所有bean实例负责。例如，如果部署了一个ShoppingCart会话bean，容器会创建一个ShoppingCart bean Home。这个ShoppingCart Home会负责ShoppingCart bean的所有实例。换句话说，如果有2000个客户，每个客户都想得到自己的ShoppingCart bean引用（要记住，这表示ShoppingCart bean组件接口的一个引用），ShoppingCart Home（惟一的那个ShoppingCart Home）就要分发2000个引用。

如果在一个应用中部署了3个bean，比如说，一个ShoppingCart、一个Customer和一个Product，那么服务器上就会有3个Home，分别表示所部署的各个bean。Home对象分发出多少个EJBObject和桩并不影响Home，Home只会有3个。

那么，这是不是说，对于每个Home，实现该bean类型Home接口的类只有一个实例呢？虽然没有必要这样，不过我们现在就是要这样想。实际上我们会把Home称为Home对象，这里假设每个部署的bean类型都只有一个实例对象。规范也可以保证这一点，你完全可以这样考虑，而不管你的开发商怎么具体完成Home实现。




**每个部署的会话bean和实体bean都有一个Home。例如，AdviceBean有一个AdviceBean Home。不管有多少个客户要得到一个AdviceBean，AdviceBean Home只会有一个。**

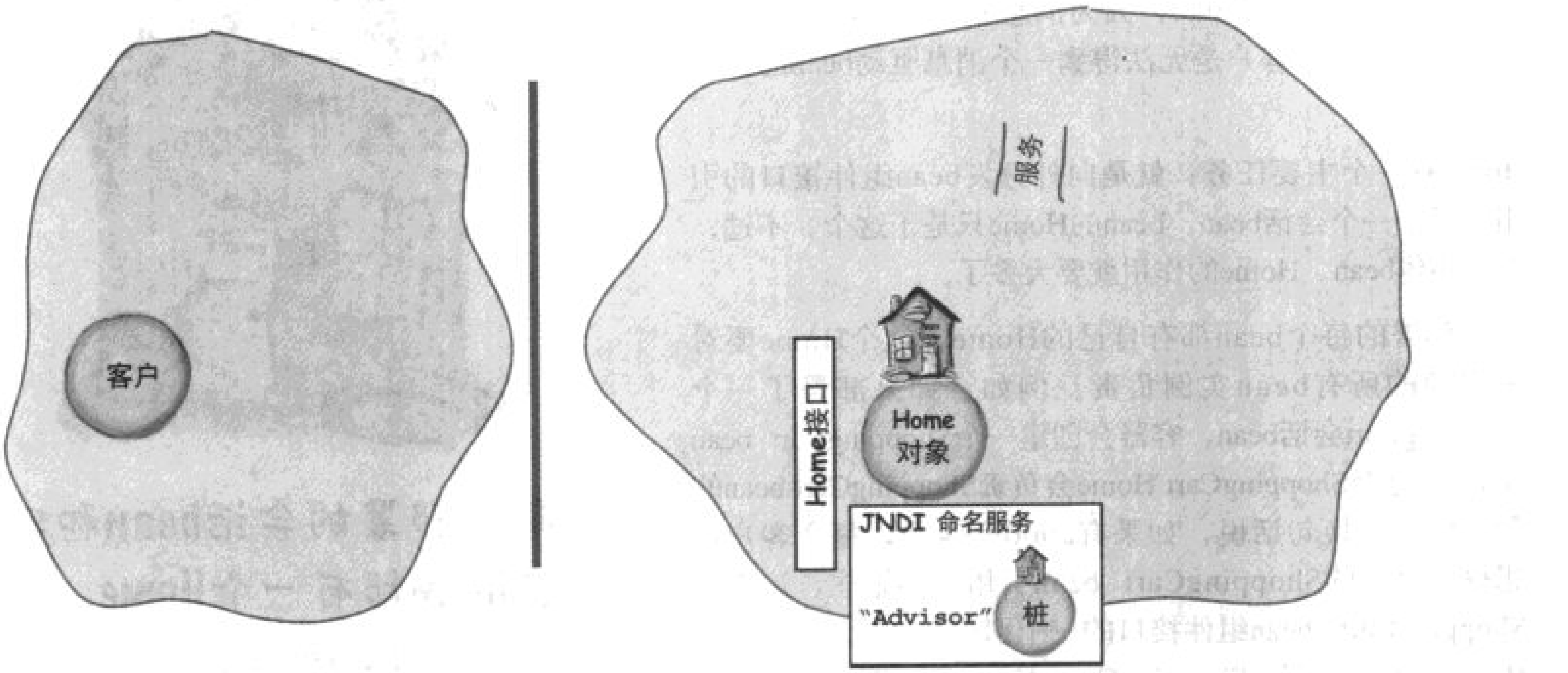
**Home的任务是分发该bean组件接口的引用。**

(从技术上讲，还不只这么简单，因为一个bean可能同时有一个本地Home和一个远程home，不过这种情况极其少见。即使出现了这种情况，每种Home（远程home或本地Home）也只有一个，而无论该类型的bean有多少个。）

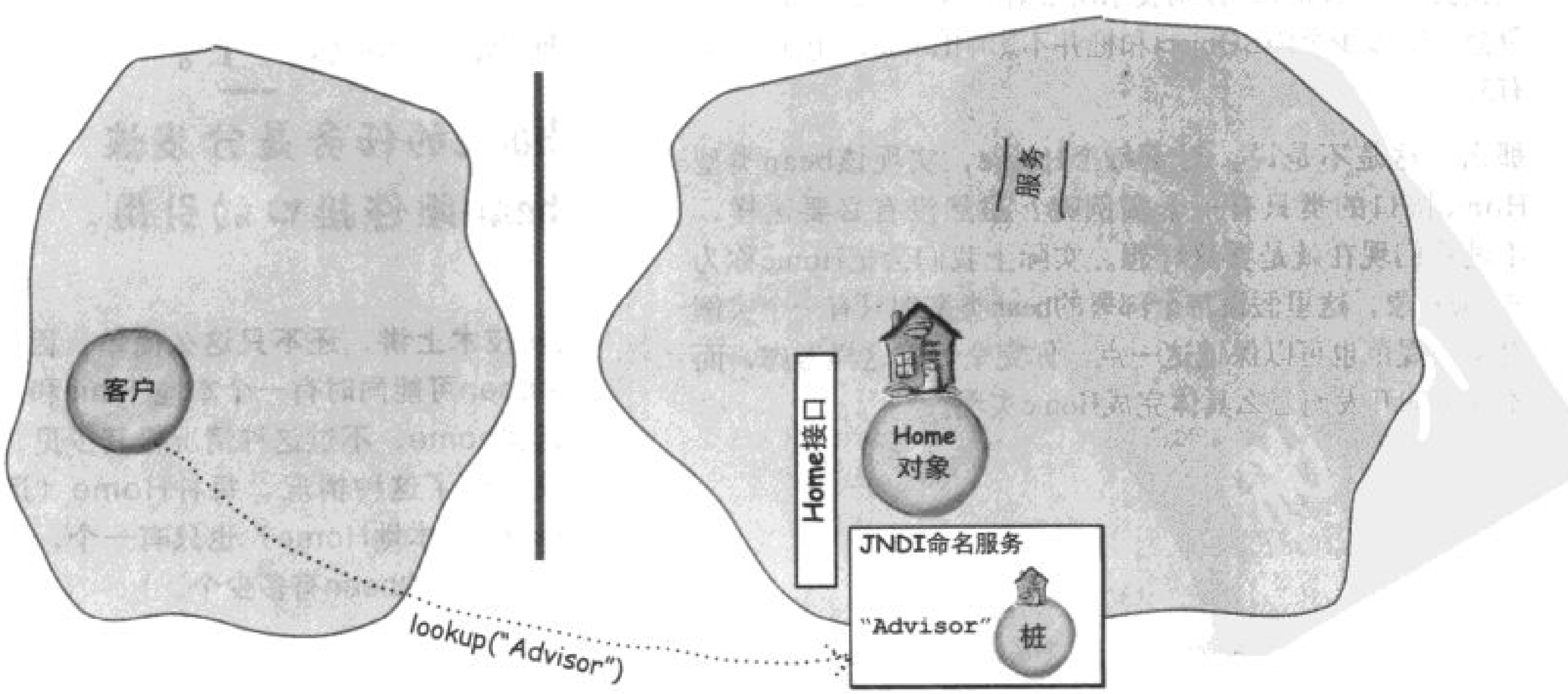
## 获得和使用AdviceBean的一个Home

（这里假设AdviceBean是一个有状态会话bean。）

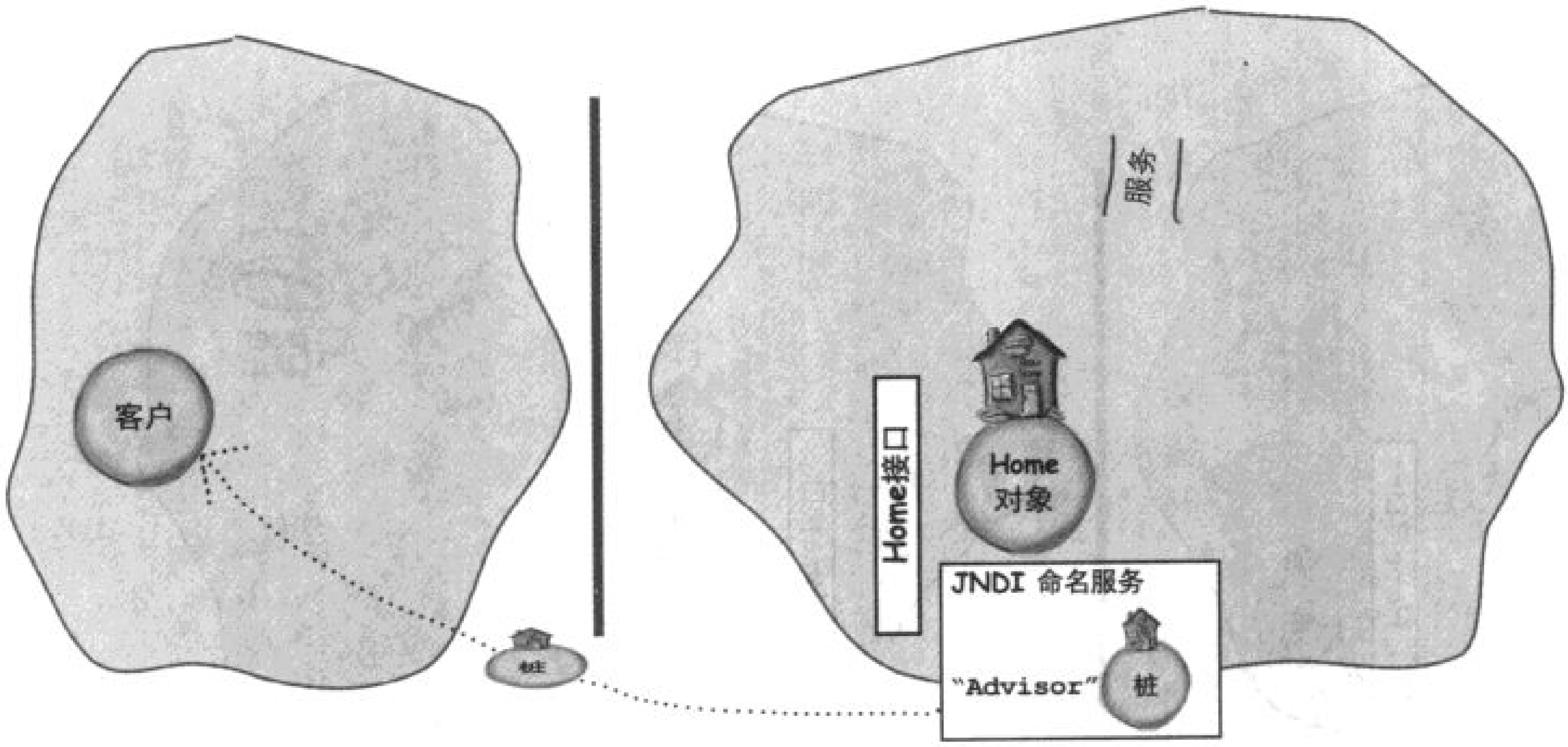
① AdviceBean得到部署，服务器实例化一个AdviceBean Home 对象，并向JNDI注册该Home对象。




② 客户对Home完成一个JNDI查找，这里使用前面注册的名字“Advisor”。

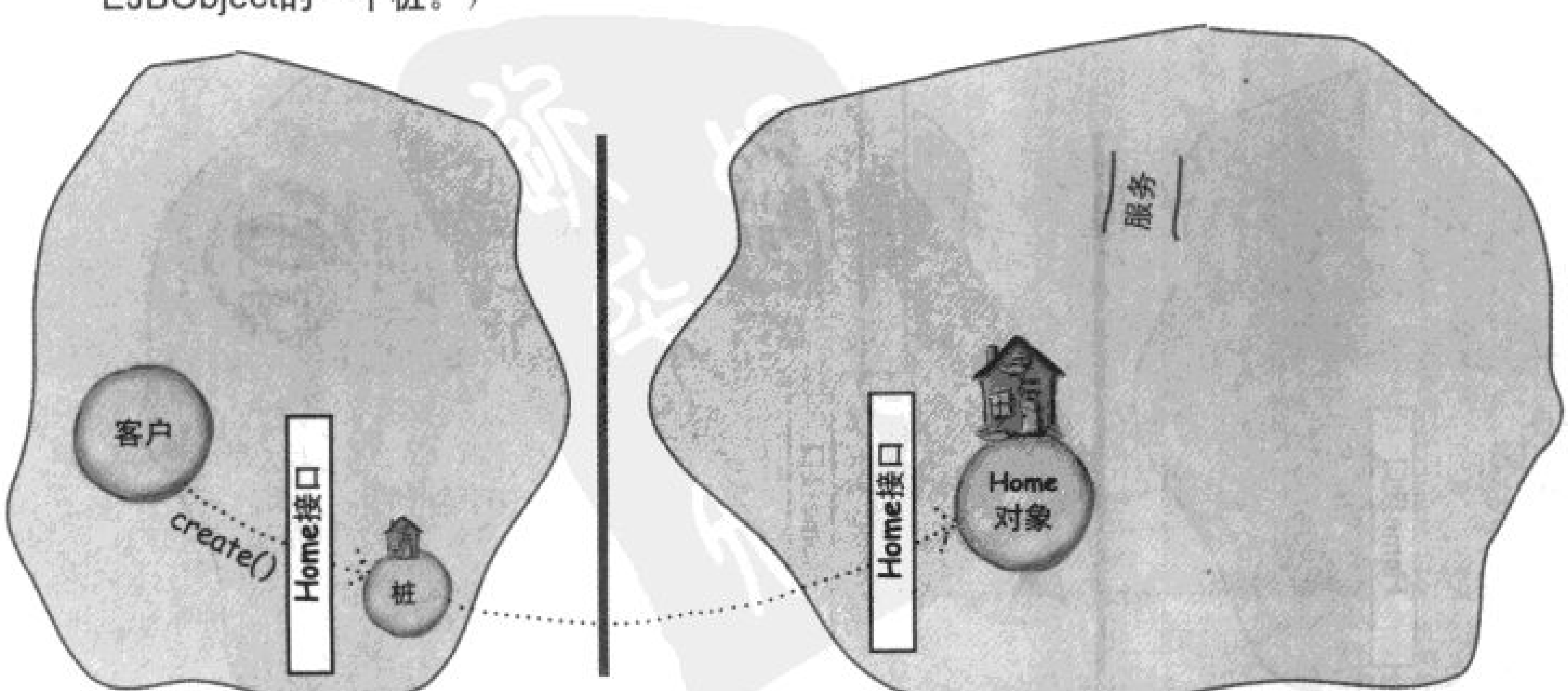


③ JNDI发回远程Home对象的一个桩。



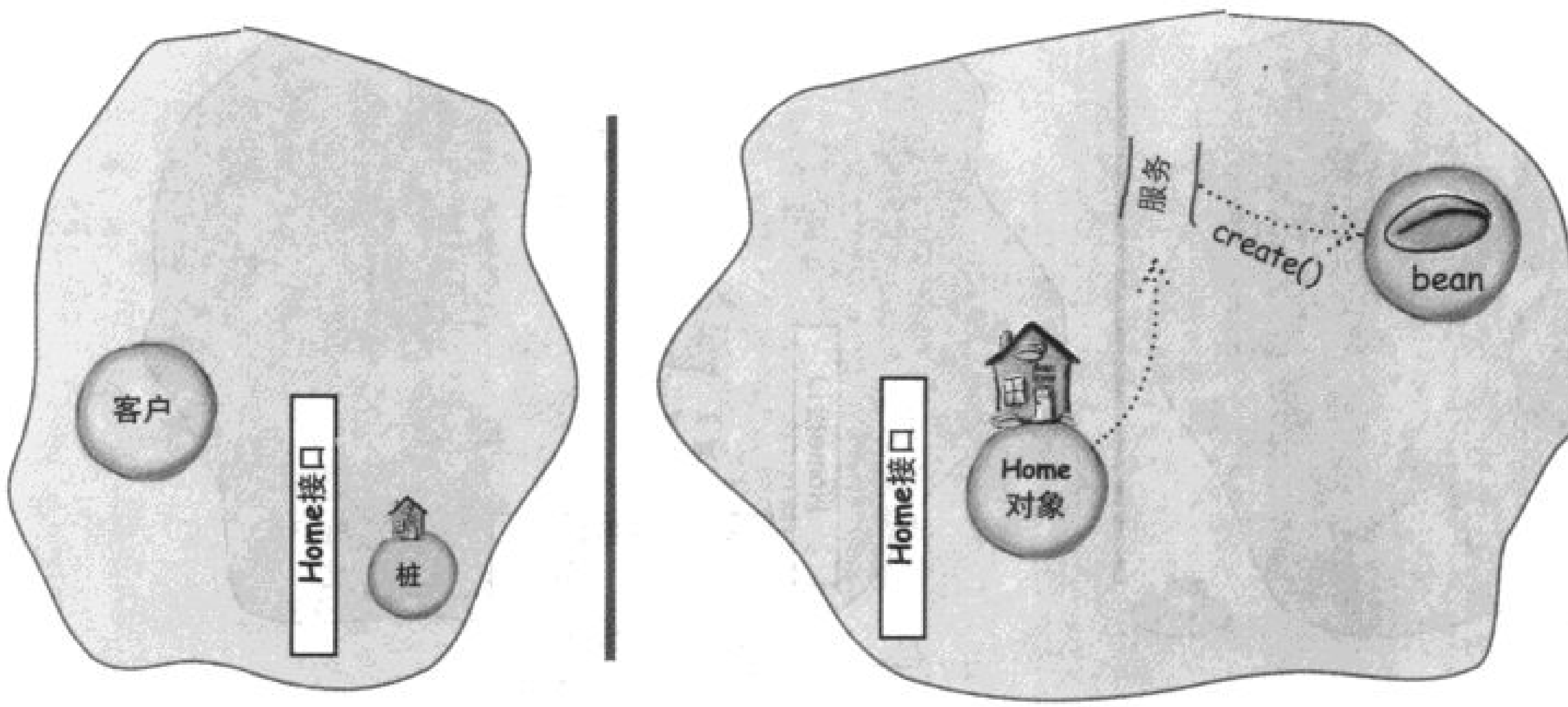

④ 客户通过调用create()向Home请求组件接口的一个引用。

（换句话说，客户想"创建"一个bean，并得到该bean EJBObject的一个桩。）

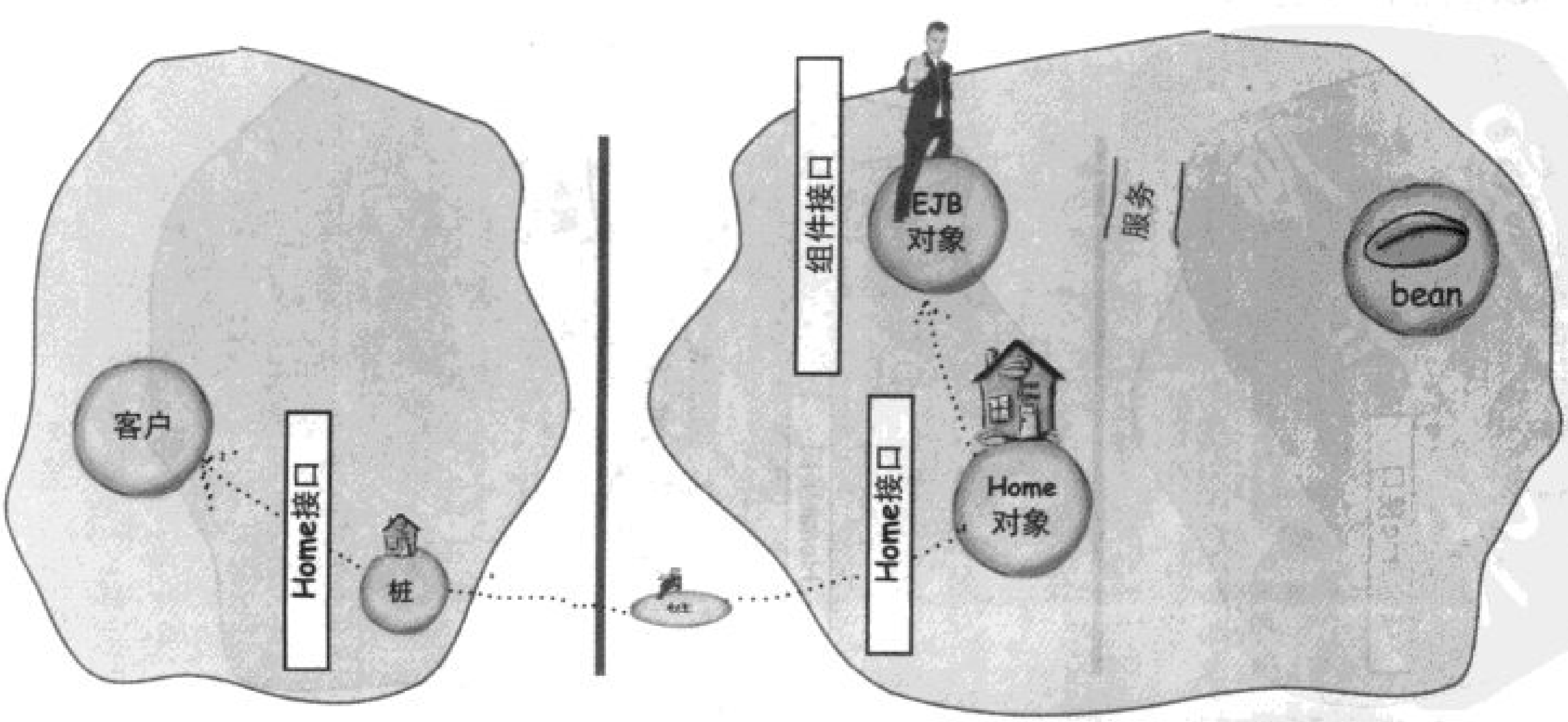


## Home对象建立EJBObject，并发回它的桩

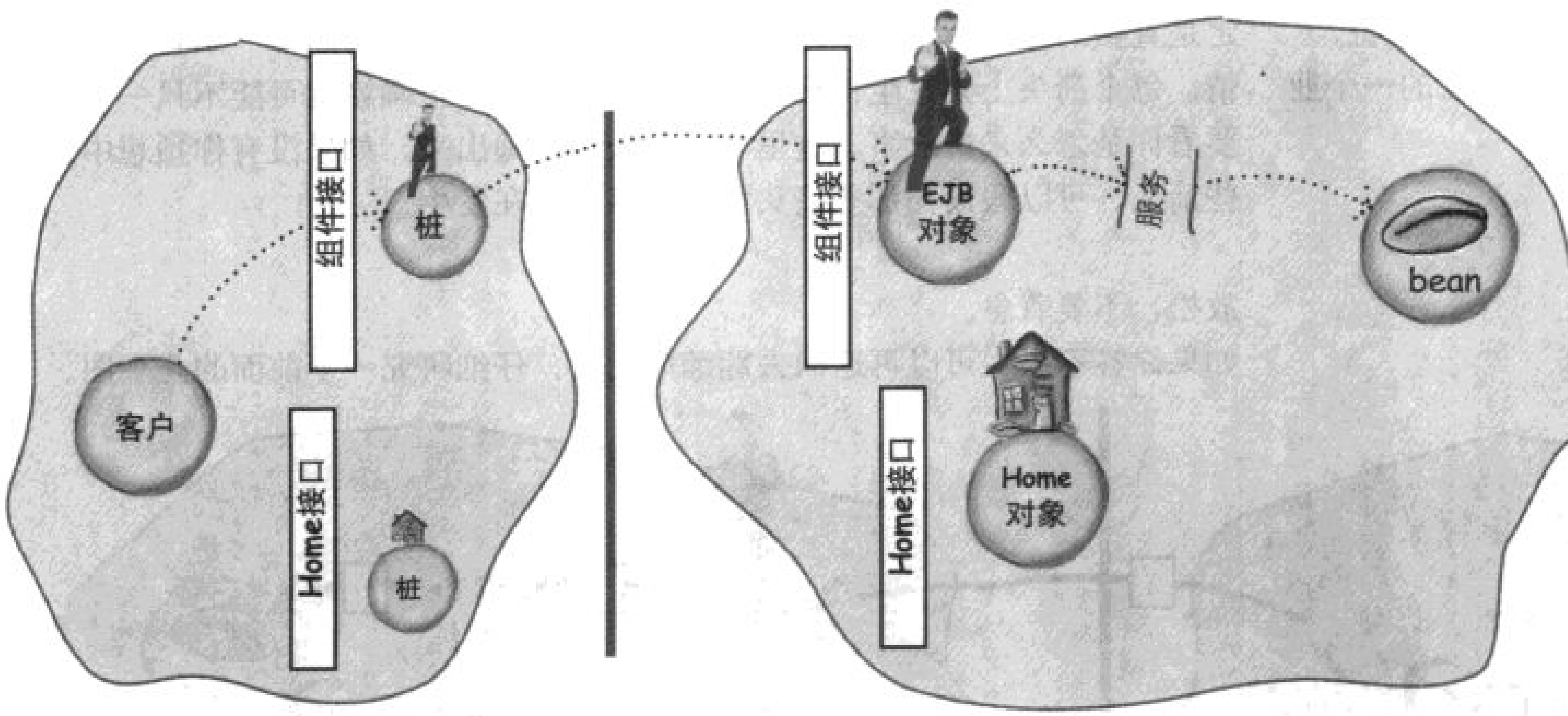
⑤ 现在“服务”介入，bean得到创建。



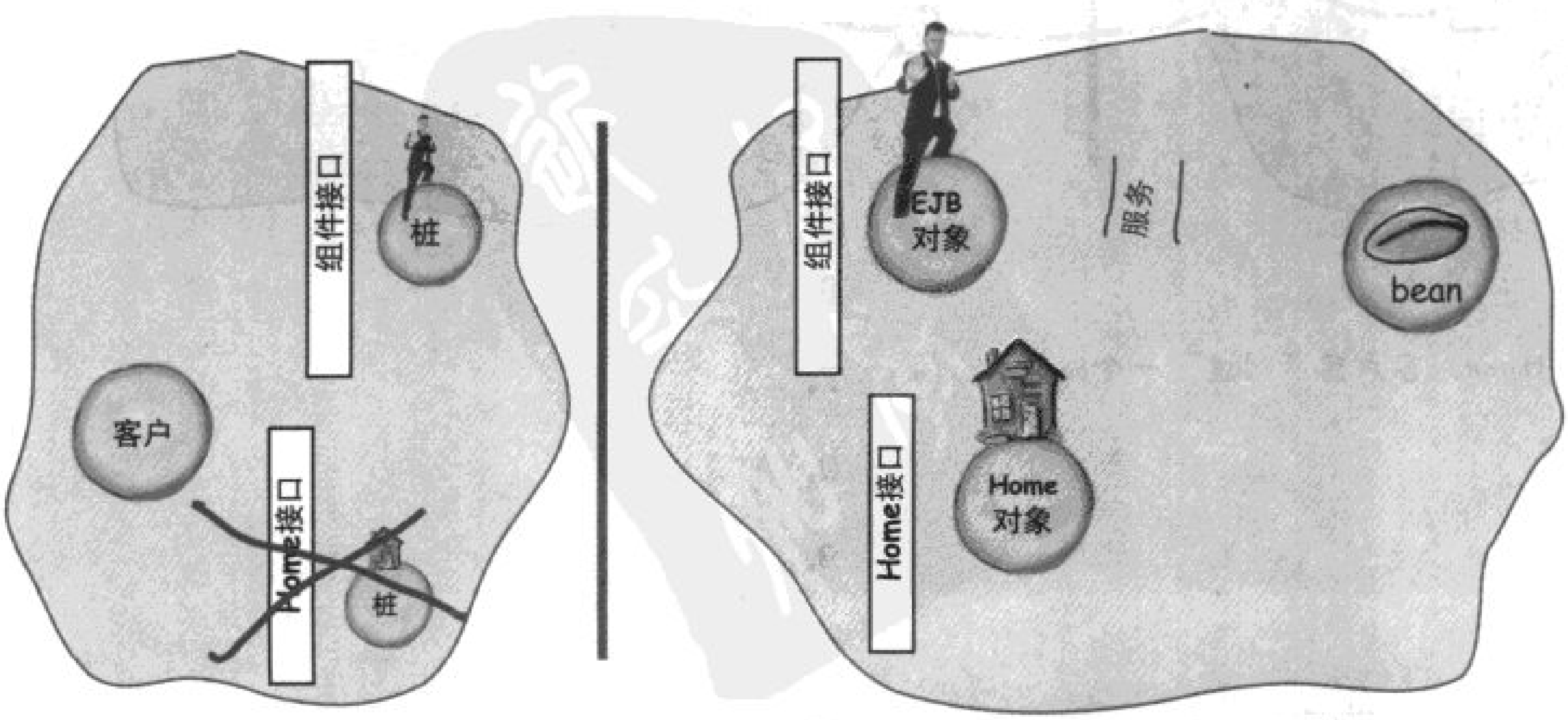

⑥ 建立EJBObject（新创建bean的保镖），它的桩返回给客户。



⑦ 现在，客户可以（终于可以）做他想做的事情了，这就是在bean上调用一个业务方法！（当然，这必须通过组件接口。）




⑧ 客户不想访问该类（AdviceBean）的更多bean时，可以删除他的Home桩，不过即使删除了Home桩，仍然可以继续调用组件接口上的方法。



**Sharpen your pencil**

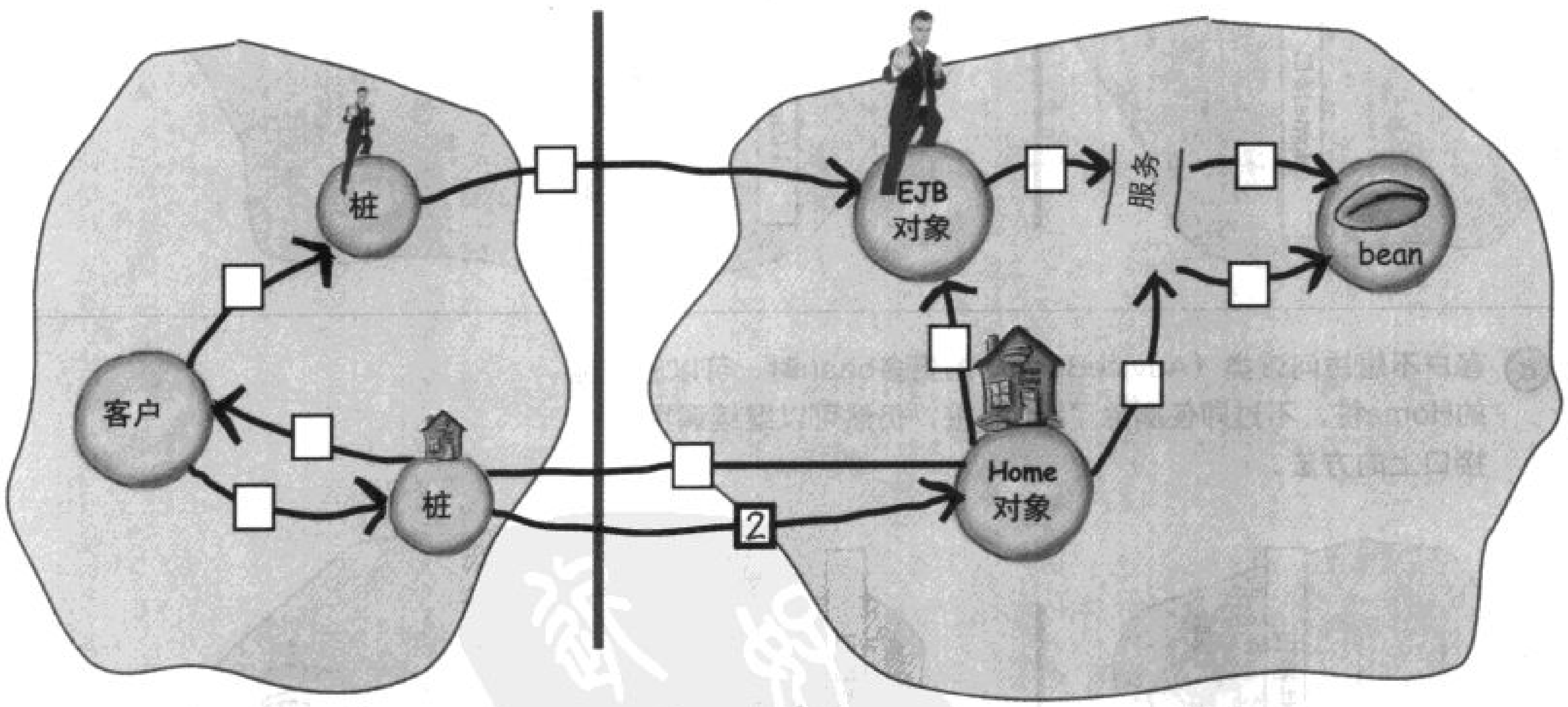
EJB生命周期:

**客户只有一个Home桩，却想调用bean上的一个业务方法。**

以下假设客户已经完成了一个JDNI查找，并得到了远程home对象的一个桩。图中的所有事件都发生在客户得到Home桩之后，现在客户想得到EJBObject的一个引用，并最终能调用该bean的一个业务方法。

按箭头出现的顺序，给箭头编上号（填在箭头上的方框里）。这些箭头不一定是直接的方法调用（不过也有可能是），它们只是指向下一个要发生的事情。每个箭头上会发生什么，请就此讲个故事。正确答案可能不只一个，这要看你是怎么来讲故事。这里有些箭头没有画出来，如果没有你预想中的某些箭头，可以假设与那些箭头相关的事情正在发生。

放松，不要着急。
如果你答不出，可以再返回去翻翻前几页，仔细研究一下前面的几个图。




1.

2. 桩告诉Home，客户想“创建”一个bean。

3.

4.

5.

6.

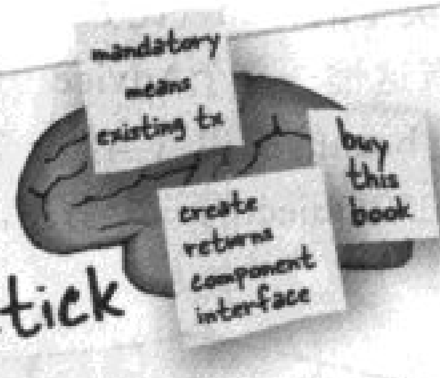
7.

8.

9.

10.

11.




**Make it Stick**

第一，
远程接口要由你来写。

第二，
服务器会建立EJBObject。

第三，
bean类本身绝不是远程的（实现远程接口）。

（好了，我们不能完完全全都列出来。你自己总结看看。记住，如果是你自己想出来的，肯定会记得更牢。）

## 要点

- 提供给远程客户的bean有两个远程接口，一个是EJB-Home接口，另一个是EJBObject接口。
- 远程接口必须扩展（直接或间接）java.rmi.Remote，而且所有方法都必须声明一个java.rmi.RemoteException异常。
- 在EJB中，扩展EJBObject的接口称为远程组件接口。业务方法都在这里声明。
- 客户不会在bean本身上调用方法，因为bean不是一个远程对象。
- 容器构建一个实现了远程组件接口的类，以此实现远程组件接口。这个类用于建立bean的EJBObject（bean的保镖）。

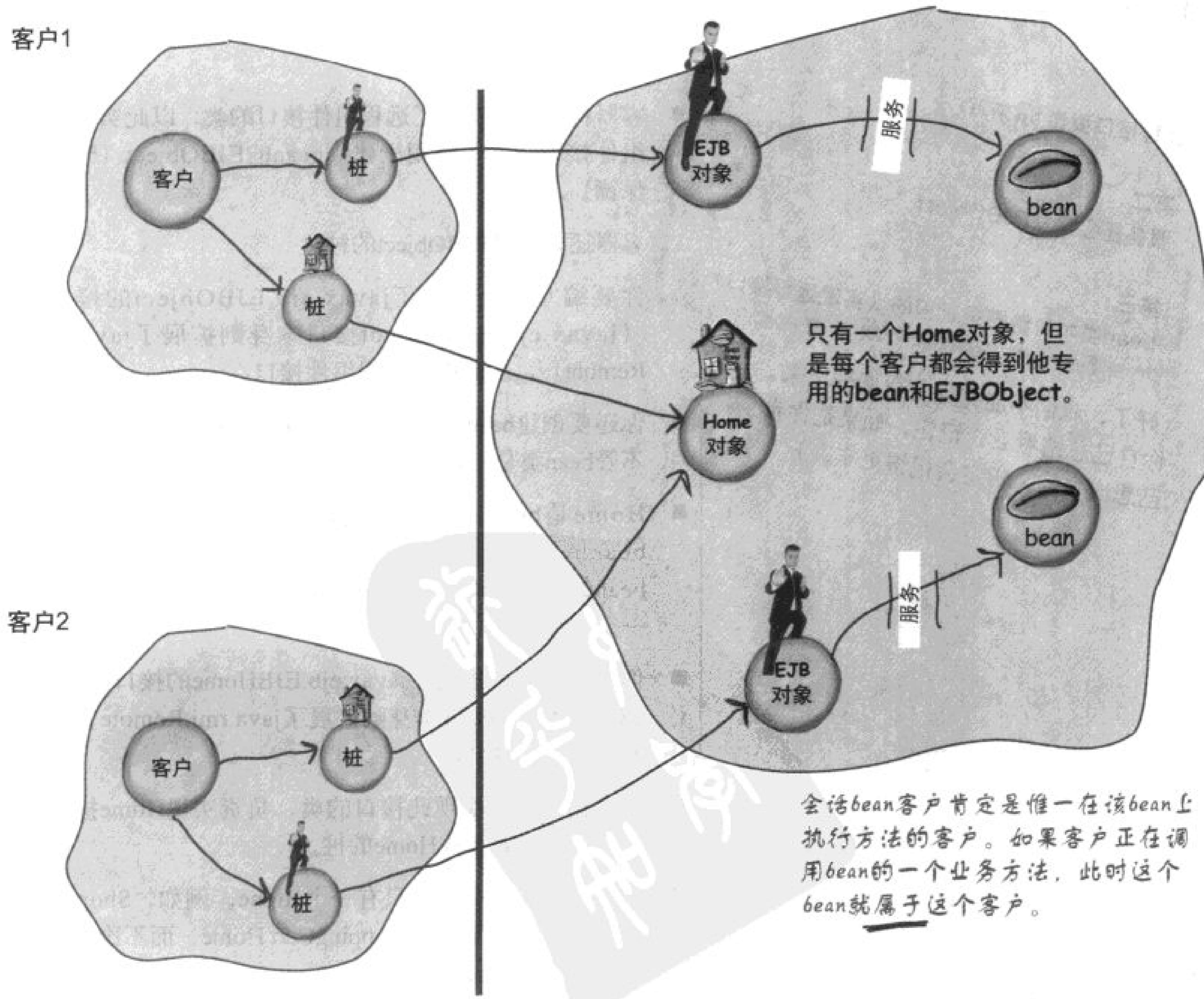
  容器还要创建EJBObject的桩。

  你要编写一个扩展了javax.ejb.EJBObject的接口（javax.ejb.EJBObject接口本身则扩展了java.rmi.Remote），从而创建远程组件接口。
- 你还要创建bean类，实际的业务方法要在这里实现（而不管bean类是不是在语法上实现了远程组件接口）。
- Home是bean的工厂。它的主要任务是向客户分发bean的引用。不过，要记住，客户并不会真正得到bean的引用，客户最多也只能得到bean的EJBObject的一个引用。
- 你要编写一个扩展了javax.ejb.EJBHome的接口（javax.ejb.EJBHome接口本身则扩展了java.rmi.Remote），从而创建Home接口。
- 容器构建一个实现此接口的类，负责实现Home接口，另外容器还要生成Home的桩。
- 对于部署的每个bean只有一个Home。例如，Shopping-Cart bean就只有一个ShoppingCart Home，而不论已经创建了多少ShoppingCart bean。

# 体系结构概述：会话bean

## 客户共享Home，但是不会共享bean.

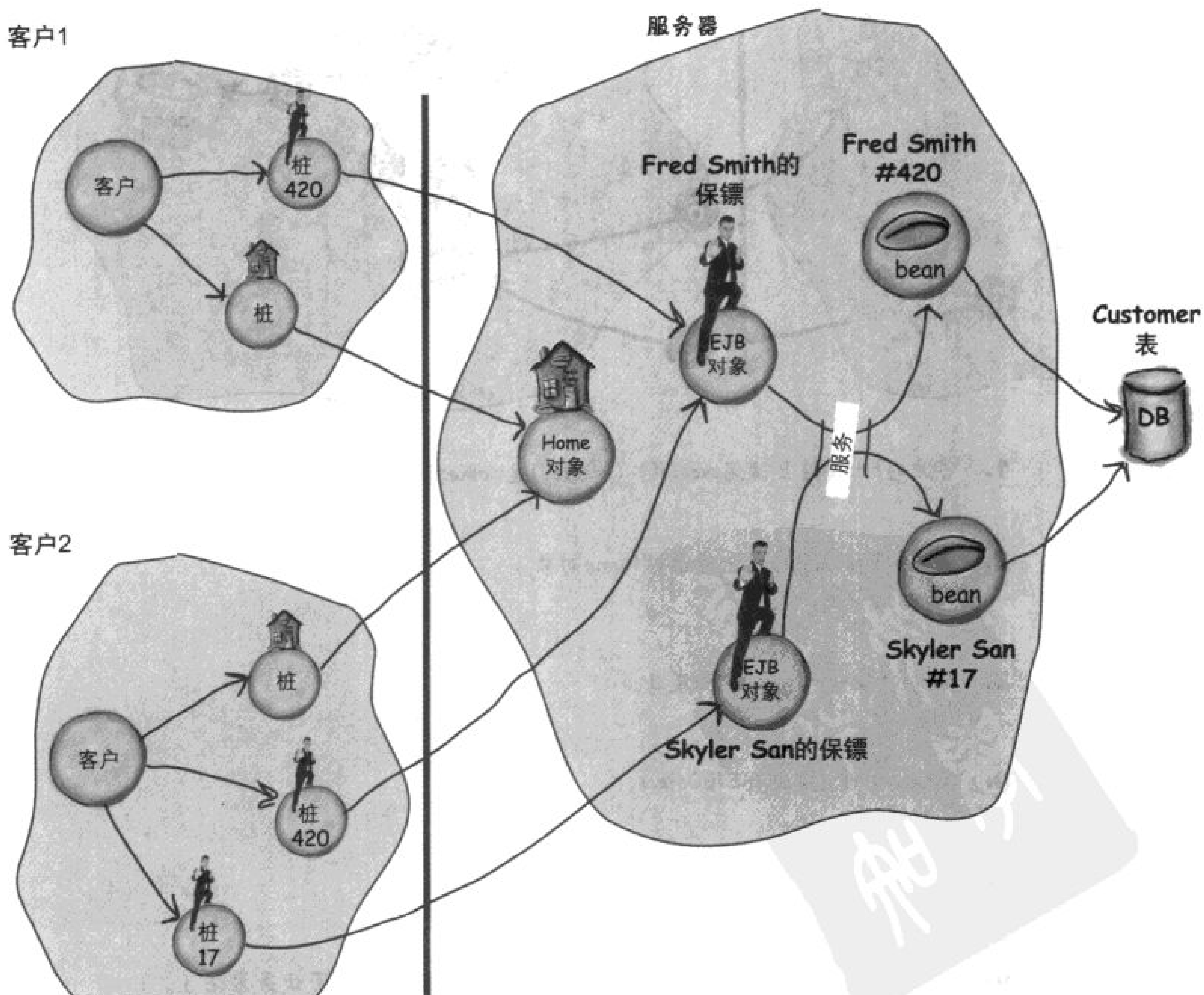
每个客户都要得到他自己的EJBObject引用和bean。客户不会和其他客户共享同一个bean，尽管根据bean是有状态还是无状态，“共享”一词有不同的含义（这个内容将在下一章介绍），但是对于这种特定类型的bean（比如AdviceBean），Home对象只有一个，所以两个不同的客户都有这惟一一个Advice Home的桩。两个客户都向Advice Home请求一个Advice bean的引用（当然，客户绝对不会得到bean实例的引用，他拿到的只是bean的EJBObject的一个引用。而且，由于EJBObject是远程的，所以客户会得到一个桩。）



# 体系结构概述：实体bean

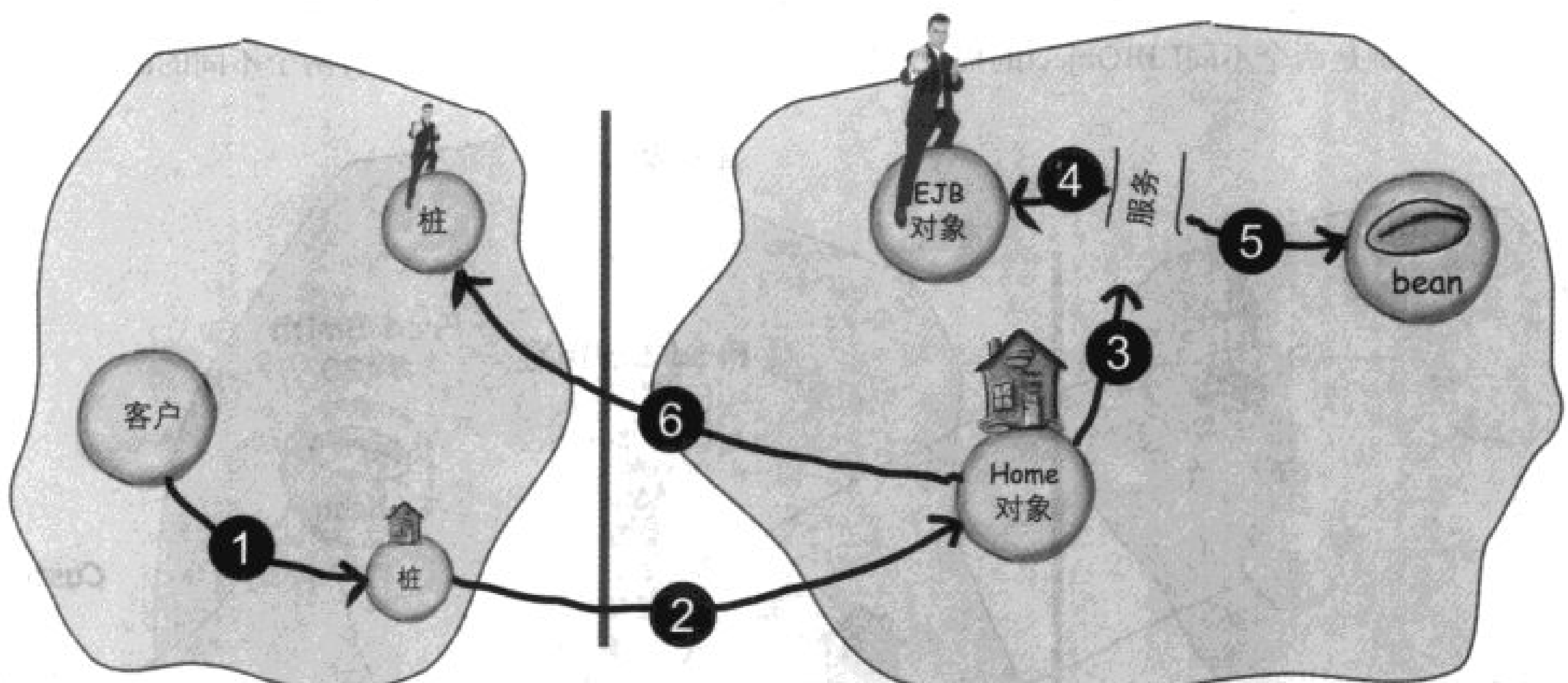
## 客户共享Home，而且可能共享bean。

对于这个bean（如CustomerBean）惟一的Home，每个客户都有他自己的一个Home引用。不过，如果两个客户想要访问同一个Customer (Fred Smith #420)，那么这两个客户都会有同一个EJBObject的引用，也就是EJBObject #420的引用。换句话说，EJBObject是一个特定Customer (比如Fred Smith）的保镖。如果所有客户都想访问Fred Smith #420，那当然各自都有自己的桩，不过所有桩都与同一个远程EJBObject通信，而且只有一个表示Fred Smith #420的bean。如果一个客户想访问两个不同的顾客，那这个客户就会有两个桩，这些桩分别是两个不同EJBObject的桩，分别对应各个顾客。这也就意味着有两个不同的bean。



# 体系结构概述：创建有状态会话bean

得到Home桩之后，客户在Home上调用create（创建方法）。Home则创建bean，而且会为bean创建EJBObject，并返回EJBObject桩。




1. 客户在Home桩上调用create()（create()是Home接口上的一个方法）。

2. 桩把create()调用发送给远程Home对象。

3. Home对象介入，并增加服务。

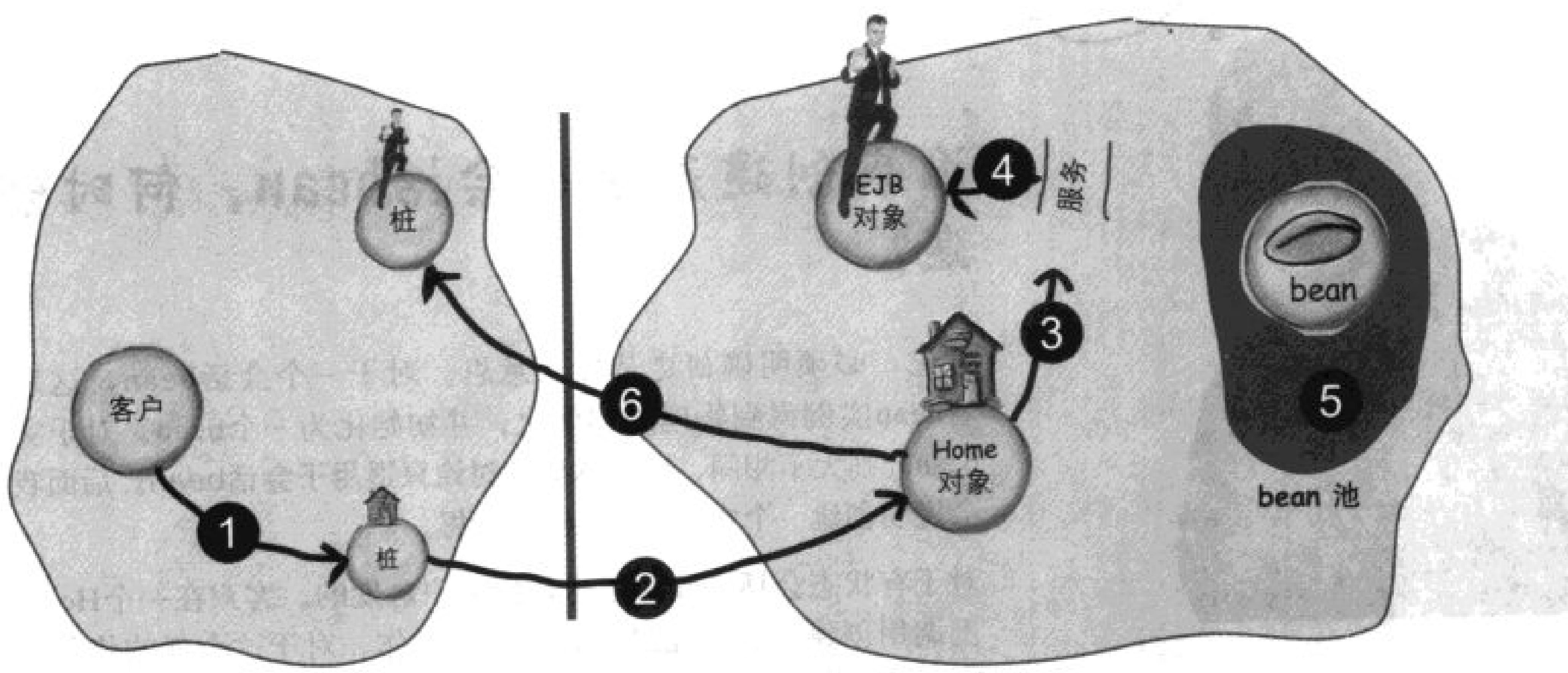
4. 为bean创建/实例化EJBObject。

5. bean本身得到实例化。

6. EJBObject桩返回给客户，这样客户就能在组件接口上调用业务方法了。

# 体系结构概述：创建无状态会话bean

得到一个Home桩之后，客户在Home上调用create（创建方法）。Home为客户提供一个现有EJBObject的桩，但是没有为这个EJBObject关联bean！相反，bean一直待在池中，直到客户使用EJBObject桩真正调用一个业务方法时，bean才会出来。




1. 客户在Home桩上调用create()（create()是Home接口上的一个方法）。

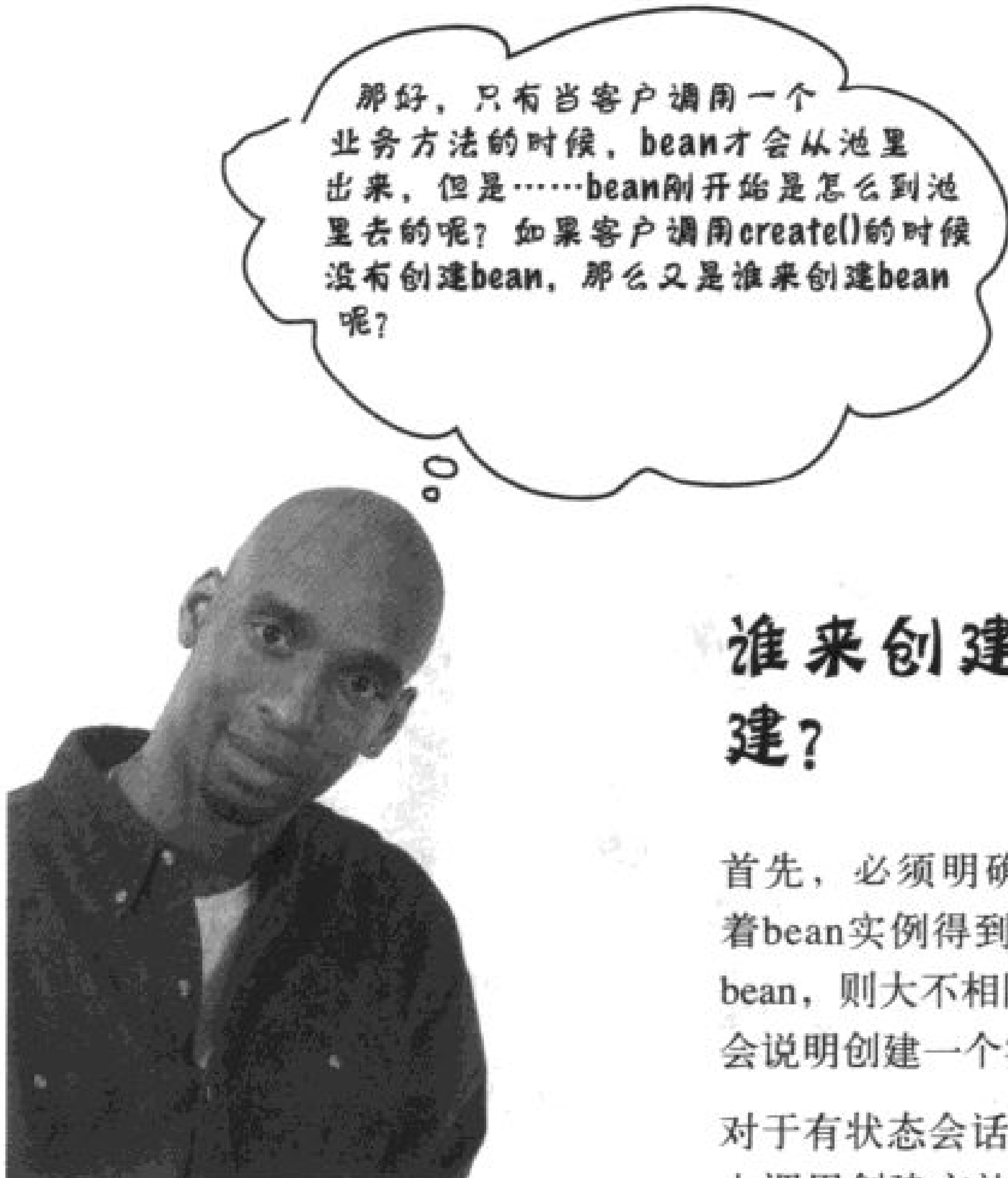
2. 桩把create()调用发送给远程Home对象。

3. Home对象介入，增加服务。

4. 为这个客户创建一个EJBObject。

5. bean一直待在bean池里！只有要为一个实际的业务方法提供服务时（客户在EJBObject桩上调用了一个业务方法），bean才会出来。

6. EJBObject桩返回给客户，这样客户就可以在组件接口上调用业务方法了。




## 谁来创建无状态会话bean，何时创建？

首先，必须明确创建是什么意思。对于一个会话bean，这意味着bean实例得到物理的实例化，并初始化为一个bean。对于实体bean，则大不相同，所以**这里的讨论只适用于会话bean**。后面我们会说明创建一个实体bean是什么意思。

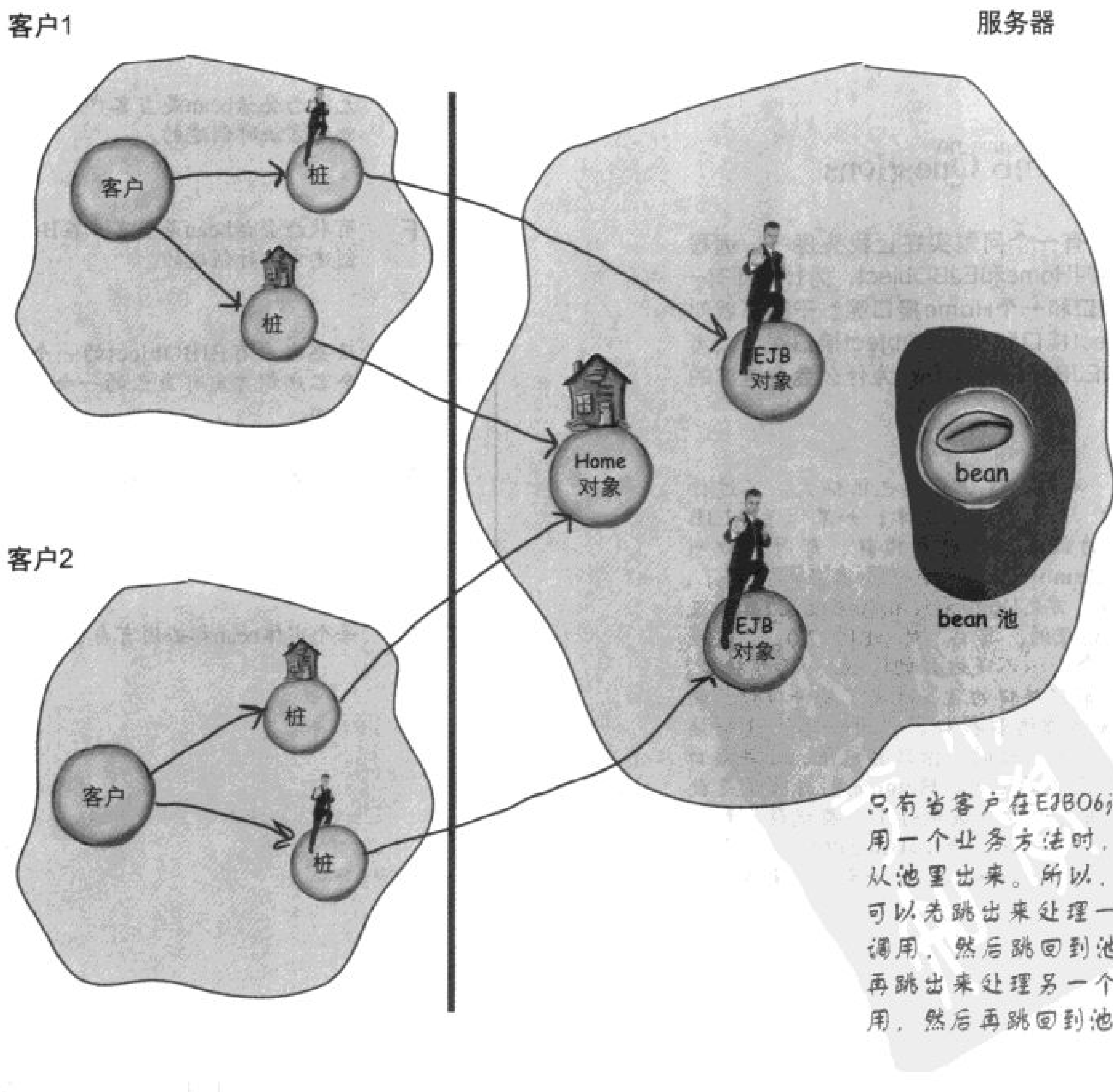
对于有状态会话bean，创建是由客户触发的。客户在一个Home桩上调用创建方法，此时会完成所有工作，对于这个新的将创建的bean，会为它实例化一个EJBObject，然后bean本身得到创建，并与EJBObject（这个bean的保镖）关联起来。

但是对于无状态会话bean，客户调用创建方法与bean的实际创建是分离的。换句话说，客户在Home桩上调用创建方法并不能说明这个时候会创建一个bean。

在容器认为需要之前，并不会创建无状态会话bean，不过这完全取决于容器。例如，容器可能在有客户请求bean（在Home桩上调用创建方法）之前先建立一堆bean实例（也就是说，创建一些bean），把它们放到一个池中。或者，容器也可能会即时地建立bean，在客户调用业务方法之前一直等着，直到有业务方法时才物理地创建bean。

# 无状态会话bean更可扩展

客户不共享EJBObject，但是同一个bean可以为多个EJBObject提供服务，只不过不能同时为它们服务。一个bean可以处理多个客户的调用，只要一次只有一个客户在做业务方法调用即可。




只有当客户在EJBObject桩上调用一个业务方法时，bean才会从池里出来。所以，一个bean可以先跳出来处理一个客户的调用，然后跳回到池中，然后再跳出来处理另一个客户的调用，然后再跳回到池中……

## BRAIN POWER

为什么采用“池”体系结构让无状态会话bean更可扩展，而不用于有状态会话bean呢？为什么不让有状态会话bean也使用bean池呢？

## there are no Dumb Questions

**问：有一个问题实在让我头疼……远程对象明明叫Home和EJBObject，为什么却有一个组件接口和一个Home接口呢？干嘛不就叫EJBObject接口和HomeObject接口呢？或者Home和EJB接口也行呀？为什么会有这样的不一致？**

**答：** 不错，这让我们也很恼火。不过你会习惯的。而且你应该庆幸，如果你学过EJB 2.0以前的版本，情况更糟糕，那时接口叫Home和Remote。这就有一个真正的问题了，这是因为，首先Home和Remote从java.rmi意义上讲都是远程的。其次，对于EJB 2.0，你甚至可以有一个……不是远程的Home和远程接口。所以，他们把接口的名字改成了组件接口，而不是直接叫作远程接口，因为实际上这个接口有可能不是远程的。你只要记住，组件接口是放业务方法的地方，Home呢，自然就是放Home方法的地方，这就行了。要记住，组件 = 业务 = EJBObject（或 EJBLocalObject，但是这个内容我们不再深入，下一章还会更多地介绍）。

## Sharpen your pencil

T F 有状态会话bean可以在多个客户间共享。

T F 实体bean可以在多个客户间共享，只要所共享的实体一样就行。

T F 无状态会话bean是当客户在Home上调用创建方法时创建的。

T F 有状态会话bean是当客户在Home上调用创建方法时创建的。

T F 只要客户有EJBObject的一个引用，每个客户都肯定有自己的一个无状态会话bean。

T F 只要客户正在对bean做一个业务方法调用，每个客户就肯定有属于自己的一个无状态会话bean。

T F 每个实体bean都必须有自己的EJBObject。

答案：F T F T F T T

## there are no Dumb Questions

**问：这是怎么工作的？是一个bean池放所有的bean，还是每一种bean分别有一个bean池？**

**答：** 实际上，我们并不知道容器实现是什么，不过，从概念上讲，应该是对应每种bean有一个池。所以，如果你部署了一个AdviceBean和一个WeatherBean，而且它们都是无状态会话bean，那它们都有自己的bean池。

**问：每个无状态会话bean都有自己的EJBObject吗？**

**答：** 不完全是这样。在对它还没有方法调用之前，无状态会话bean不需要保镖。所以，客户得到一个EJBObject引用，但是bean并没有和这个EJBObject关联，直到客户真正调用一个业务方法为止。这个时候，bean会从池里出来，为方法提供服务。所以，客户得到的EJBObject对应的是一种bean（AdviceBean、WeatherBean等等），而不是针对一个bean的特定实例。

**问：为什么有状态会话bean没有池？**

**答：** 你考虑前一页上的"开动脑筋"了吗？如果没有，还没有得出你自己的想法，那么先别急着往下读。如果你已经考虑了，我们认为你已经知道了答案，而我们只是确认你的答案对不对……

要记住，有状态bean维护着客户的会话状态。这说明，bean必须在多个客户方法调用之间保存客户的特定状态（换句话说，它必须记住有关这个客户的情况）。

再来看一个购物车，这是一个有状态bean，每次客户调用addItemToCart()时，它需要记住客户的购物车里有些什么。另一方面，无状态bean则不用记住有关客户的任何信息，所以对于客户来说，有一个特定类型的无状态bean就行了，而不管到底是哪一个。

如果AdviceBean只是返回一条建议，而这条建议与以前告诉客户的建议（或者以前客户告诉它的信息）没有任何关联，那么AdviceBean就没有必要作为一个有状态bean。在这种情况下，客户每次在组件接口（即EJBObject）上调用getAdvice()时，任何AdviceBean都可以运行这个方法。

另一方面，如果对AdviceBean有所修改，比如让它返回随机但不重复的建议，那么这个AdviceBean就必须是有状态的，这样才能跟踪以前的建议，而不至于重复。

**问：有状态bean会把客户特定的状态保存多长时间？**

**答：** 只在会话期间保留。会话会一直持续，直到客户告诉bean他的工作已经完成（在bean的组件接口上调用remove()），或者服务器瘫痪，再或者是bean已经超时（我们将在会话生命周期一章介绍超时的内容）。

**问：那有状态会话bean是不是不可扩展的？**

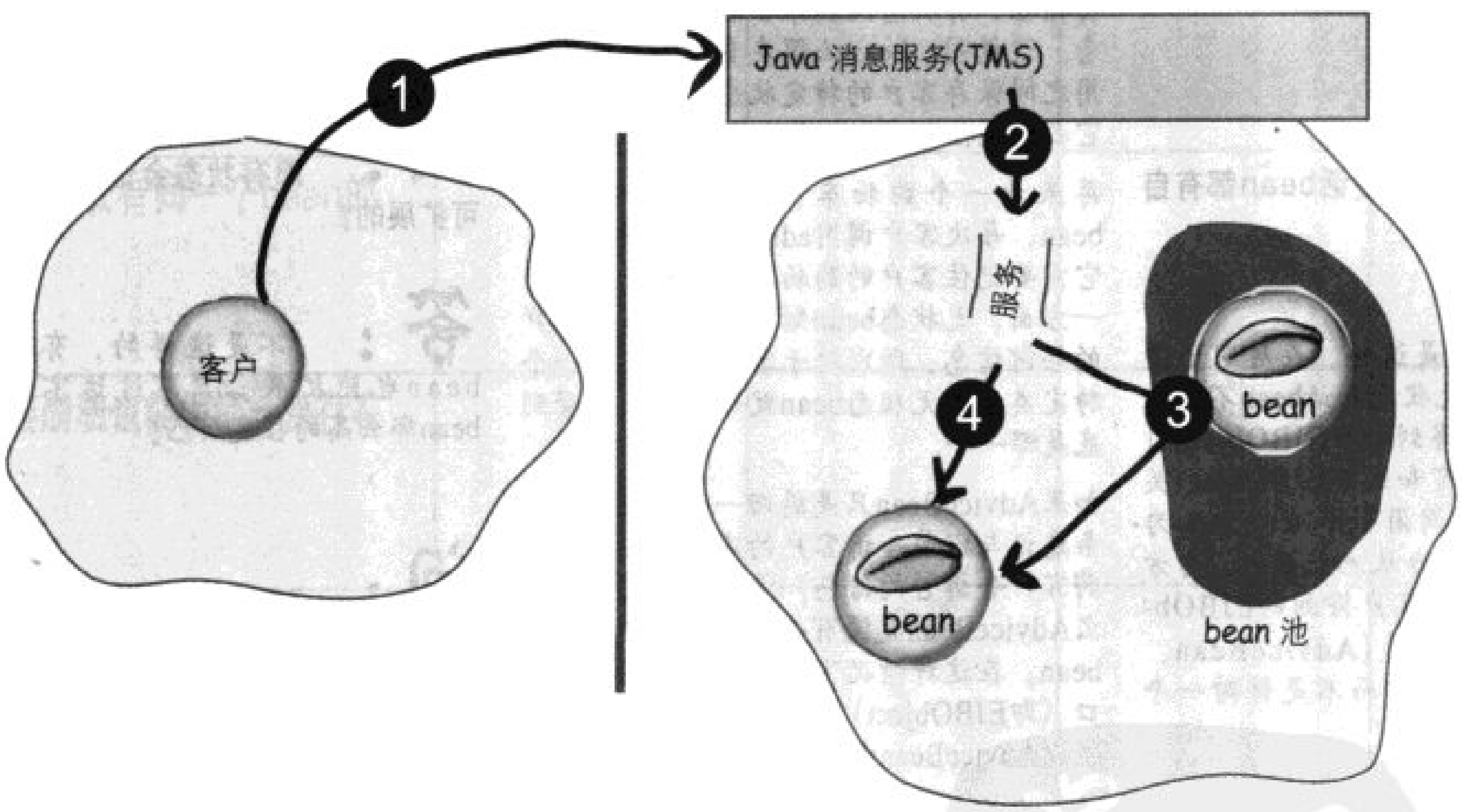
**答：** 不是这样的，有状态会话bean也能扩展。只不过达不到无状态bean那么高的程度而已。

**问：如果每个客户总是需要一个bean，你怎么让一个有状态bean做到可扩展呢？**

**答：** 你确实需要为每个客户分配一个单独的bean，但是并不是每一个bean都会大量消耗资源。如果有状态会话bean客户在方法调用之间要耽搁很久，可以把有状态bean临时地拿下，置于一种称为钝化的状态中。这种状态下，当然还会保留客户特定的状态，但是可以减少当前在服务器中活动的bean数量。客户调用一个业务方法时，这个bean会从钝化状态中出来，再回到活动（激活）状态下。

# 体系结构概述：消息驱动bean

消息驱动bean没有客户视图。这说明，消息驱动bean没有向客户展现方法的接口（远程或本地）。换句话说，消息驱动bean没有Home或EJBObject，它们没有Home接口或组件接口。




1. 客户向一个JMS消息服务发送一条消息。

2. 消息服务把消息传送给容器。

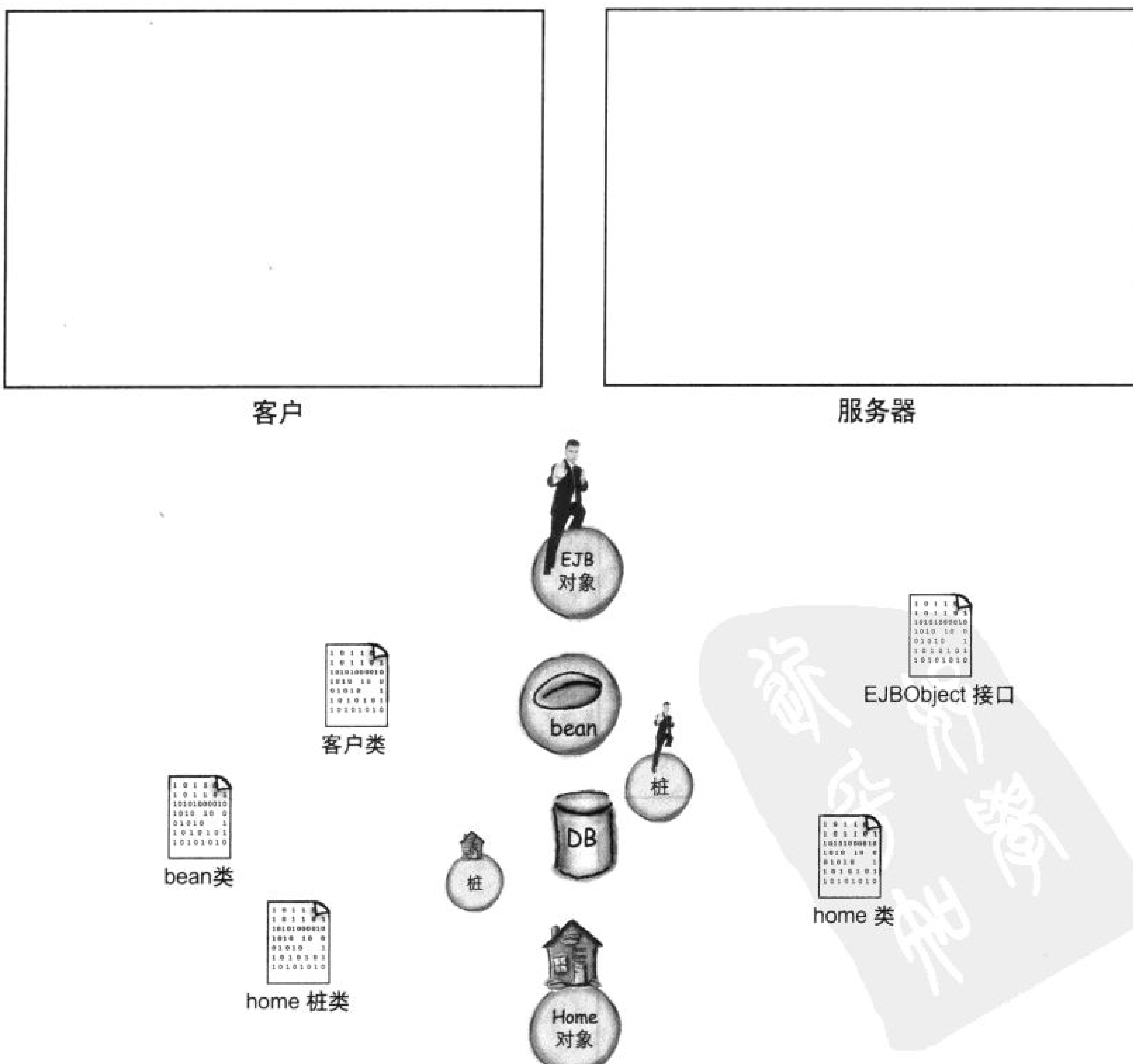
3. 容器把一个消息驱动bean从池中拿出来。

4. 容器把消息传送给bean（调用bean的onMessage() MessageListener 接口方法）。




# 哪归哪？

把对象和类放到合适的位置上，可能在客户上，或者在服务器上，还有可能在客户和服务器上都有（不错，对象是可以重用的）。注意：这里并没有把所有参与的部分都画出来，所以等你做完后，如果认为还有一些东西（类或对象）应该在这个图中出现，可以把它们也画出来！





练习

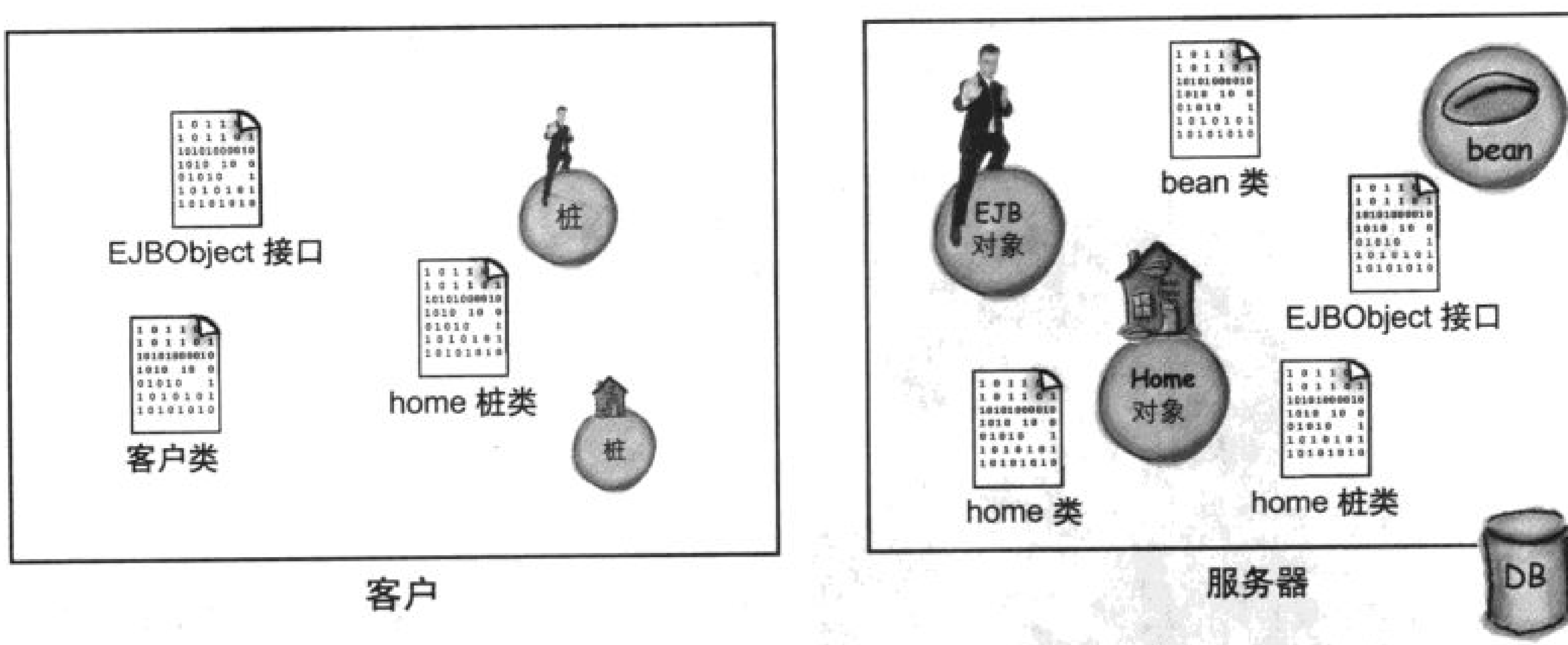
# bean的组织

填表，对应标签，在有相应特性的bean类型上打勾（加上说明就更好了）。我们已经完成了一个空。如果你做不出来，可以回顾一下前两章。有些东西你可能还要猜。那没关系，等学完本书，你肯定能全部做出来。我们相信你，你肯定行的。[提示音乐起……]

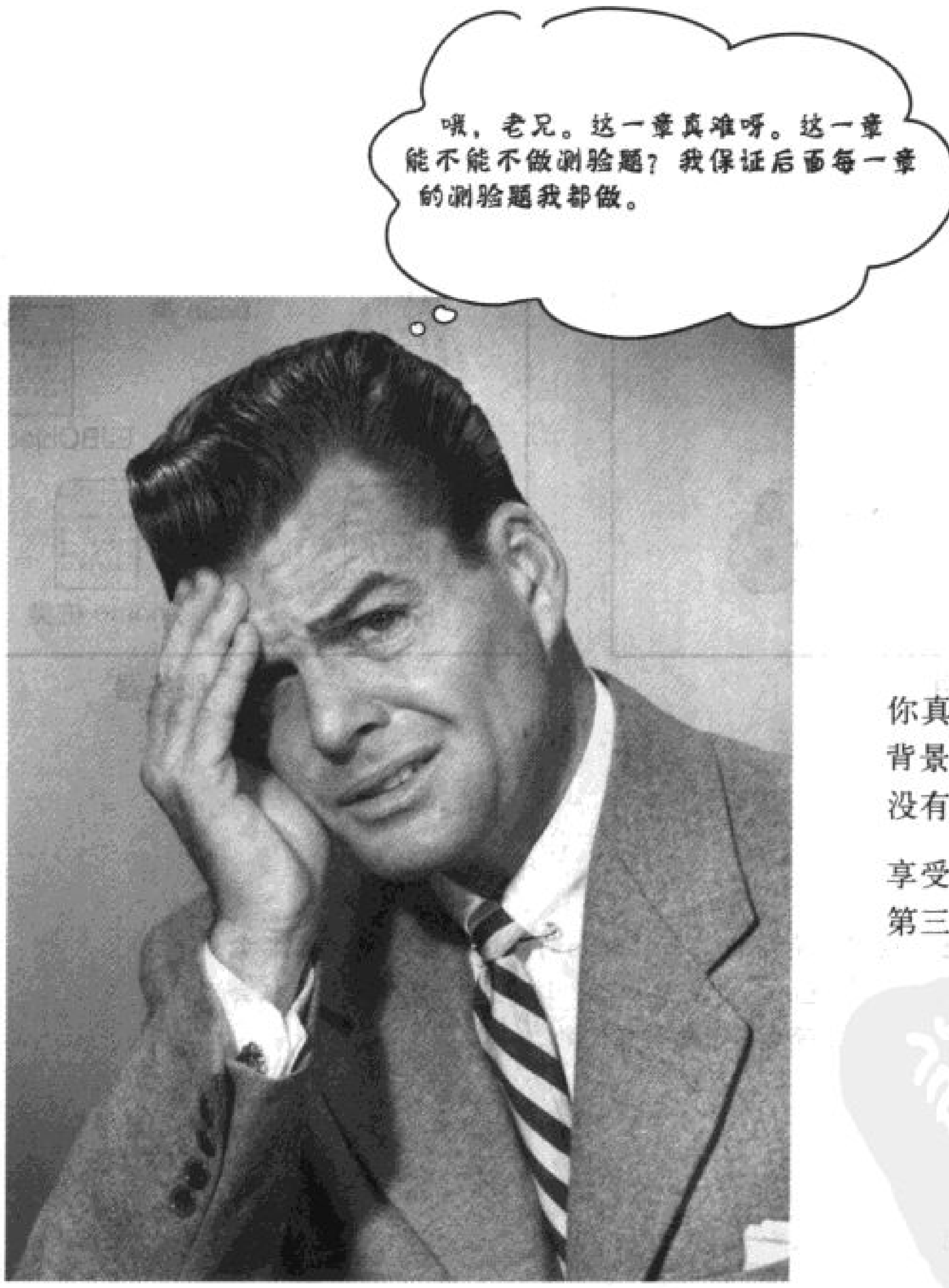
| | 无状态会话 bean | 有状态会话 bean | 实体 bean | 消息驱动 bean |
|---|---|---|---|---|
| 使用一个池 | 是，因为它们不保留任何客户特定的数据，不需要每个客户都有一个bean。 | | | |
| 多个客户可以有同一个bean的引用 | | | | |
| 即使容器崩溃也能保证一直存在 | | | | |
| 有一个客户视图 | | | | |
| 允许异步通信 | | | | |
| 表示一个过程 | | | | |
| 表示底层持久存储库（如数据库）里的一个“东西” | | | | |

练习答案

## 哪归哪？




注意：这里没有给出前面“bean的组织”练习的最后答案。我们希望**你自己**把它填出来。这是一个特殊的学习机会，为此你以后会带着感激想起我们的。




你真幸运。这一章讲的都是些背景知识，所以这里确确实实没有任何考试要求和测验题。

享受这个轻松时刻吧，要知道第三章只有“一页之遥”……

# 3 客户视图

# 表现你自己



**不能让bean太隐密了。** 客户需要看到你有什么（消息驱动bean除外，它没有客户视图）。Advice Bean在它的组件接口里展现了getAdvice()方法，组件接口就是声明业务方法的地方。不过，这并不是客户看到的全部。要记住，Advice接口扩展了EJBObject，这个EJBObject接口又有它自己的方法。这些方法客户也能看到，而且能调用。Home接口也一样。在这一章中，我们将学习到底会向客户展现什么，以及客户如何工作，这包括远程接口和本地接口。

# 客户视图

| 标准大纲： | | 具体含义： |
|---|---|---|
| 2.1 | 对会话bean的本地和远程home接口，明确有关这些接口的客户视图有哪些正确和不正确的说法或例子，这包括客户查找会话bean home接口所用的代码。 | 必须了解有关home接口的所有内容。这个要求不包括实体bean home的特殊特征，不过涵盖了有关bean home客户视图的大部分细节，这些内容将在这一章介绍。<br><br>例如，必须知道哪些方法在javax.ejb.EJBHome（远程home接口）里，哪些方法在javax.ejb.EJBLocalHome（本地home接口）里。只是知道这些方法是什么还不够，你还必须知道它们会在哪些情况下调用。例如，你要知道，远程会话bean客户可以使用bean的home来删除这个bean，但是本地客户不可以。你还要知道，本地home比远程home的方法要少，还应该了解这对客户来讲有什么意义。<br>最后，对于要求2.1，必须知道客户在一个bean的home接口上完成JNDI查找的有关细节。这包括客户查找代码的语法、完成查找的规则，以及如何使用home接口来得到bean组件接口的一个引用。例如，你要知道，窄化一个home桩有哪些规则，另外，给定一个代码段，你必须能识别出客户是本地客户还是远程客户。 |
| 2.2 | 对会话bean的本地和远程组件接口，明确有关这些接口的客户视图有哪些正确和不正确的说法或例子。 | 这个要求与2.1很像，只不过它关于的是组件接口。不过同样地，你必须知道javax.ejb.EJBObject和javax.ejb.EJBLocalObject中的所有方法，还要知道客户怎么使用这些方法，另外给定一段代码，你必须能够做到：通过查看代码就能找出远程客户和本地客户之间的区别。 |

# 客户到底想要什么

客户有一个目标、一个梦想、一个追求。她想在bean上调用一个业务方法！业务方法是组件接口中展现的方法。不要忘了这个最终目标，否则很容易陷在细节的泥淖中不能自拔，因而迷失了方向。不过，如果一直都关注着客户的需求，就会更容易记住一些事情，比如会话bean home中create()方法的返回类型是什么。



# 一切都从home接口开始




客户想要的是bean。可惜呀，客户永远也得不到bean，因为没有人（除了容器）能跟bean直接对话。客户最多能得到bean的保镖（组件接口）的一个引用。客户要调用bean home接口上的一个方法来得到bean组件接口的一个引用。

**客户实际上想要的是bean的一个引用。但是，客户最多只能得到bean的组件接口（EJB对象）*的一个引用。**（译者注：本书中交替出现了EJB对象（EJB object）和EJBObject，这二者是相同的。）

**但是如果客户想要一个EJB对象引用，客户必须得到bean home接口的一个引用。**

**所以这就找到源头了……客户要在bean的home上完成一个查找。**

## there are no Dumb Questions

**问：** 你说“客户得到bean组件接口的一个引用……”这句话是什么意思？Java中不能有接口的引用，可以引用一个对象，但是不能引用一个接口。引用变量可以声明是一个接口类型，但这是另一码事呀。

**答：** 嗯，实际上这是一样的。在本书里，另外在规范里，还有在考试中，到处都能看到这种说法，“引用一个接口”，你可以在心里把它替换成“引用一个实现了此接口的对象。”

如果你以前没有见过采用这种约定的文献，那看上去可能有点奇怪，不过在学到第四章之前，你最好熟悉并习惯这种说法。真正习惯了之后，你反而会奇怪怎么会有人认为这两种说法不同呢？……

**问：** 你说客户的最终目标是在bean上调用方法（不错，实际上是在bean的组件接口上调用方法，不过我想你知道我的意思）。不过，对于实体bean，可以在home中放业务方法，对吧？所以对于实体bean，有时客户的目标就是使用home，这样说对吗？

**答：** 没错，你是对的。这些业务方法称为“home业务方法”（与普通的“home方法”或普通的“业务方法”相区别）。但是这是一种特殊情况，我们将在本书后面再来讨论这种情况。为什么可能只需要实体bean的home，这还有其他一些原因。例如，如果你想在数据库中创建一些新顾客，但是你又不希望对这些实体顾客的引用做其他操作。

*这里用“EJB对象”一词表示bean的组件接口（保镖），也就是说，它要接收本来发给bean的方法调用，而不管客户是本地的还是远程的。

# 客户如何使用会话bean：创建、使用和删除

注意：在后面几页中，我们只讨论bean的远程接口。然后会了解一下作为特例的"本地"接口。



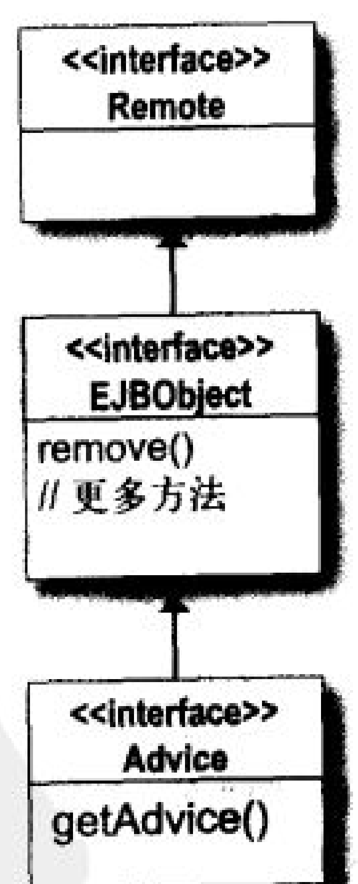

bean开发人员必须在bean的home接口中放一个create()方法。

**❶ 创建**

客户向home接口请求bean组件接口的一个引用。

(这说明，客户在home桩上调用一个create()方法。)

---




javax.ejb.EJBObject接口有一个remove()方法，可供客户调用。

在组件接口中，开发人员会放入业务方法，但是由于组件接口扩展了EJBObject接口，所以这两个接口中的方法客户都能看到。

**❷ 使用**

客户调用组件接口中声明的业务方法。

(意思是客户调用EJB对象桩上的方法。)

**❸ 删除**

客户告诉bean，他已经用完了这个bean。

(这说明客户调用bean的EJB对象桩上的remove()。)

# 首先，必须得到一个home接口引用

## 换句话说，必须得到home对象的桩……我们要用它来调用create()，这才能得到我们真正想要的东西——EJB对象桩！

你（客户）想要home接口的一个引用时，需要通过JNDI。这个过程非常简单：向JNDI提供一个逻辑名（也就是部署人员告诉服务器要用的名字），然后得到一个实现了home接口的对象。

```
Context ic = new InitialContext();

Object o = ic.lookup( Advisor );

// 其他步骤……
```

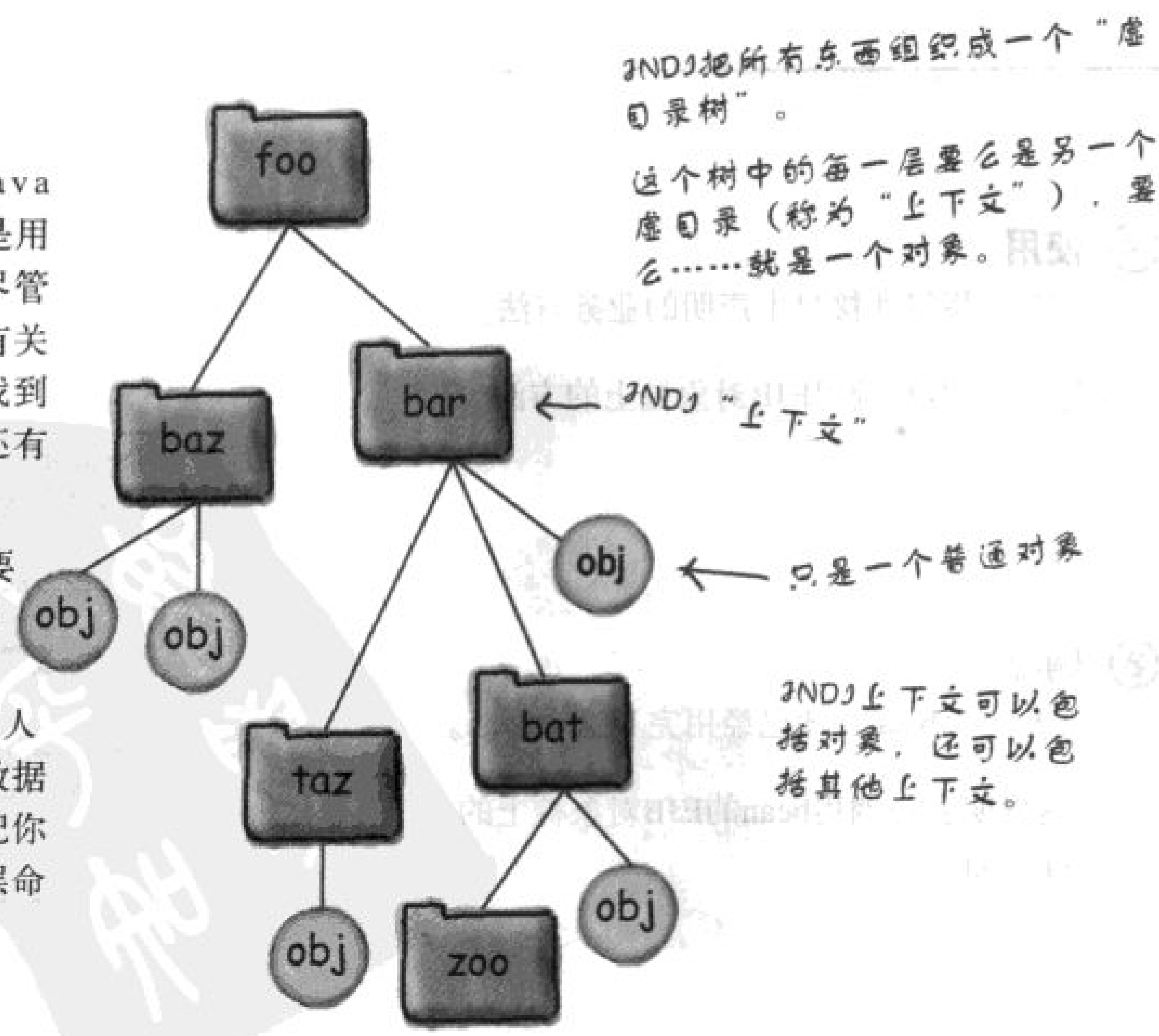
← 分配给"ic"的InitialContext对象是JNDI查找服务的一个引用。

提供一个名字（就是bean部署人员向服务器注册时用的名字），并得到一个对象。

## 什么是JNDI?

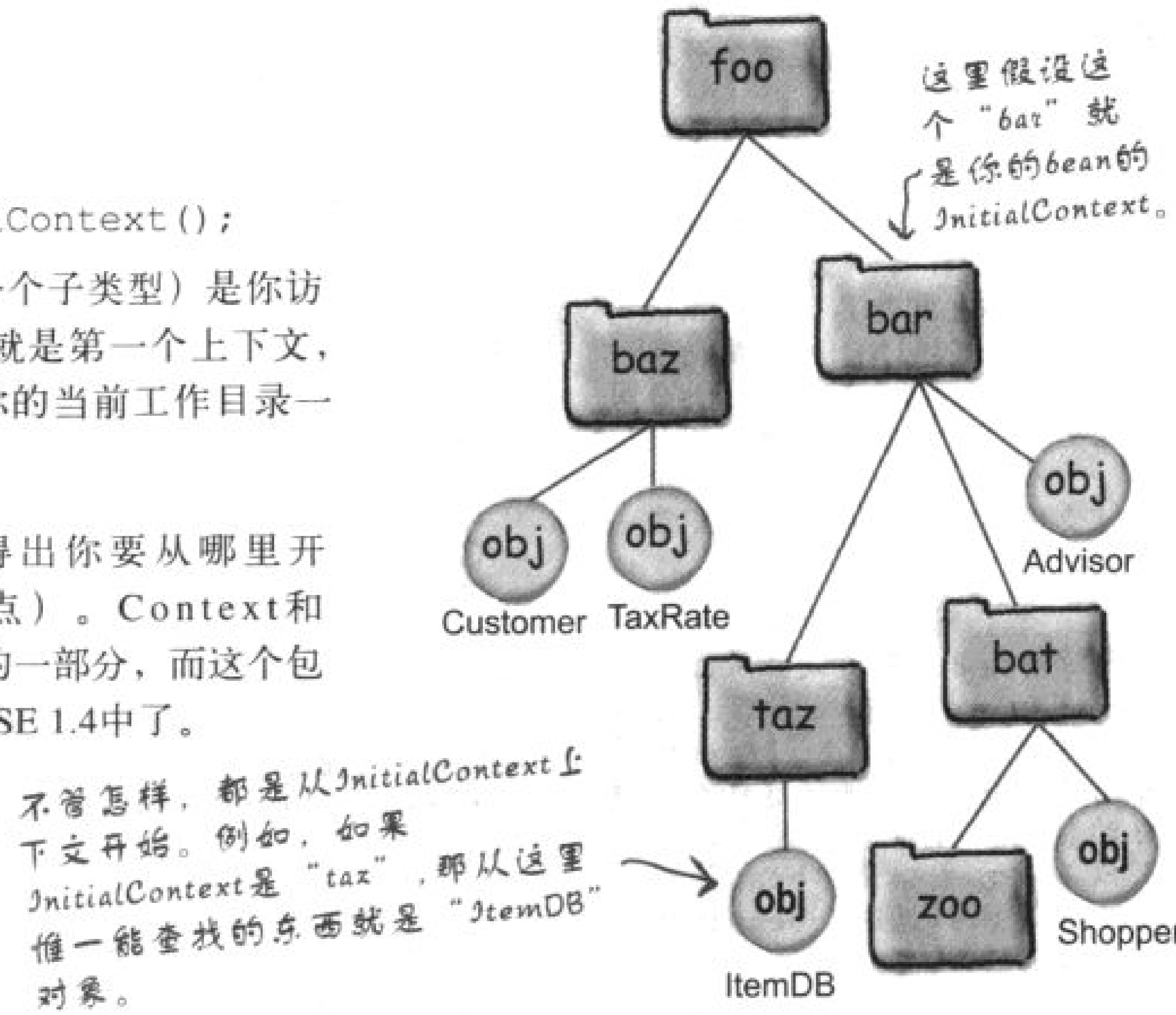
JNDI代表Java命名与目录接口（Java Naming and Directory Interface），这是用于访问命名与目录服务的一个API。尽管JNDI功能很强大，但你只需要了解它有关EJB的一点点内容，这就是客户怎么找到它，客户怎么用它，bean怎么用它，还有就是怎么把东西放到JNDI里。

JNDIAPI可以配合多种不同的服务，只要那种服务有一个JNDI驱动程序设计（这称为服务提供者，Service Provider）。它和JDBC很像，对于JDBC，你（开发人员）可以使用JDBC API向多种不同的数据库发送SQL语句。JNDI驱动程序则是把你在JNDI API上做的方法调用翻译成底层命名/目录服务能够理解的东西。



## 得到home接口桩

JNDI "虚目录树"




### ① 得到一个InitialContext

```
Context ic = new InitialContext();
```

InitialContext（这是Context的一个子类型）是你访问JNDI树的入口点。这实际上就是第一个上下文，你要从这里开始导航。就像是你的当前工作目录一样（用cd命令切换到的目录）。

InitialContext构造函数会得出你要从哪里开始（我们稍后就会谈到这一点）。Context和InitialContext是 javax.naming包的一部分，而这个包是J2EE的一部分，它也增加到J2SE 1.4中了。

### ② 使用InitialContext得到bean的home

```
Object o = ic.lookup("Advisor");
```

查找方法取一个String参数，这个参数必须与分配给该bean　JDNI部署的名字匹配。如果部署人员为bean指定了一个额外的上下文，（在部署时）把这个上下文命名为"bat/Advisor"，那么查找代码要改为：

```
ic.lookup("bat/Advisor");
```

### ③ 把查找的结果指定给home接口引用

```
AdviceHome home = (AdviceHome) o;
```

警告：这个代码不完全正确……不过看上去好像是对的。后面几页我们会分析到底哪里有问题。

上下文查找方法的返回类型是Object，所以必须把它强制转换回bean的home接口类型，然后才能调用AdviceHome方法。

## there are no Dumb Questions

**问：** **我怎么知道是哪个开发人员给bean起的名字？**

**答：** 实际上，并不是bean开发人员（在EJB中，这个角色称为bean提供者，也就是具体写bean代码的人）给bean提供JNDI名。要记住，bean提供者可能为Beans‘R’Us公司编写这个bean，让它作为一个可重用组件，所以这个bean会在哪里使用以及如何使用，对此bean提供者可能一无所知。

部署人员才会用一个逻辑名来注册bean，所谓部署人员就是让bean作为某个应用的一部分在服务器上运行的人。不过关键是，如何知道已注册bean的名字，对此并没有什么标准或自动机制。作为一个客户，必须有人以某种方式告诉你bean注册为什么名字，比如说这个bean注册为“Advisor”。

请注意，尽管“Advisor”描述了这个服务，但这个String与bean其他任何部分的名字并没有直接的对应。应该记得，组件接口是Advice，home接口是AdviceHome，而bean本身是AdviceBean。“Advisor”这个名字只是部署人员认为很好而已。

当然，在你自己的公司里，可能（而且很可能）有一些严格的命名规则，对部署时如何向JNDI注册bean有严格的要求。

（除非这是你自己的公司，在这种情况下，你当然可以为所欲为，比如用你最喜欢的摇滚明星的名字给bean取名。）

**问：** **我在考虑一个更大的问题……我怎么知道到哪去找服务器呢？另外我怎么指定服务器？我没有看到代码里有IP地址或TCP端口号呀！**

**答：** 这个问题很好。不错，是有些神秘，对不对？这个问题现在有3个答案：

（1）这里藏了点东西，我们用的代码之所以能工作，只是因为我们在使用参考实现（Reference Implementation，RI），甚至……只是因为我们是在客户所在的同一个物理主机上运行服务器。所以，我们是在利用默认配置，这些默认配置自动地已经有了，因为我们运行的是参考实现（RI）。

（2）上面的第1点也不完全对，因为如果这个代码在一个bean里，那这个代码就是正确的（这个问题会在“bean环境”一章中再做说明）。

（3）在实际中，客户确实需要知道怎么去找注册bean home的JNDI服务。对此有许多方法，可以把信息传递给InitialContext构造函数（这是一个Properties对象，其中包含InitialContext寻找服务器和开始上下文所需的所有信息）。或者，JNDI特性可以放在客户机的很多位置上。不管是哪一种情况，客户都必须得到点什么，可以是提供给InitialContext构造函数的信息，也可以是提供一个性质文件。对于每个开发商的服务器，这会有所不同，所以你要查看你的文档，了解一下客户到底需要什么。




**考试时，你不用对JNDI了解太多！**

如果考试要求与客户有关（也就是这一章的考试要求），你只需知道完成一个JNDI查找的基本过程……也就是需要从一个InitialContext开始，然后调用lookup()，这个方法会返回一个Object类型的对象。

你无需知道客户或服务器怎么查找和得到正确的InitialContext的引用，只要有一个InitialContext就行了。

在“bean环境”一章中，我们会增加一些有关JNDI的信息，说明bean自己如何使用JNDI来查找专门为这个bean放的一些东西。

不过，要求你了解的JNDI知识就这么多了。除了Context.lookup()方法外，你不用知道其他JNDI API的任何细节。

## 下面再来看看完整的客户代码

```
import javax.naming.*;
import java.rmi.*;
import javax.rmi.*;
import headfirst.*;
import javax.ejb.*;

public class AdviceClient {

   public static void main(String[] args) {
      new AdviceClient().go();
   }

   public void go() {
      try {
         Context ic = new InitialContext();
         Object o = ic.lookup( Advisor );

         AdviceHome home = (AdviceHome) PortableRemoteObject.narrow(o, AdviceHome.class);

         Advice advisor = home.create();

         System.out.println(advisor.getAdvice());

      } catch (RemoteException rex) {
         rex.printStackTrace();
      } catch (CreateException cex) {
         cex.printStackTrace();
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

一堆导入语句，在这一页的最后我们会逐个地说明。

InitialContext是访问JNDI命名服务的入口点，我们在这里根据名"Advisor"完成查找

这是什么东西？？？为什么不用普通的类型强制转换呢？

在home上调用create（创建方法），得到我们真正想要的东西（组件接口）。

这是我们的根本目的！就是要在bean上（通过EJBObject桩）调用一个业务方法。

这不是处理异常的好办法（或者说，实际上这里的做法是不处理异常）……我们只是想显示一些受查异常……

### Sharpen your pencil

看看各个类在哪个包里。一个包名可以同多个类名对应（也就是说，这里可能有多个类放在同一个包里）。

如果你不太确定，可以尽量地猜猜看。

| 包名 | 类名 |
|---|---|
| javax.naming | InitialContext |
| java.rmi | AdviceHome |
| javax.rmi | PortableRemoteObject |
| headfirst | RemoteException |
| javax.ejb | Advice |
| | CreateException |

## 你可能觉得一个简单的类型强制转换就足够了……

Context.lookup()方法的返回值类型是Object。所以我们认为一个简单的类型强制转换（cast）就足够了，这就能把o引用的对象强制转换成AdviceHome实现（我们知道它实际上就是AdviceHome）：

```
Context ic = new InitialContext();
Object o = ic.lookup("Advisor");
AdviceHome home = (AdviceHome) o;
```

看上去这是对的，实则不然。对于一个远程home接口，类型强制转换还不够。

## 但并非如此。你还必须将对象窄化！

窄化（narrowing）会强制JNDI查找返回的对象肯定（绝对）实现home接口。
换句话说，你可以把它强制转换为 AdviceHome。

```
Context ic = new InitialContext();
Object o = ic.lookup("Advisor");
AdviceHome home = (AdviceHome) PortableRemoteObject.narrow(o, AdviceHome.class);
```

## 那好，但是为什么不能只做一个普通的类型强制转换呢？

根据规范，你（客户）必须假设服务器在使用 RMI-IIOP而不是常规的普通RMI。正常的RMI使用JRMP作为通信协议，它假定我们总是按Java的方式行事。如果是普通的RMI，那你肯定知道从查找得到的结果必定是一个home接口（与home接口满足IS-A关系）。换句话说，返回的这个对象的类类型实现了该bean的home接口。在这种情况下，正常的Java类型强制转换就可以把对象转换回home接口类型，这样你就能调用home方法了！否则，要记住，虽然你想调用的是create()，但实际上只能调用Object类型的方法（equals()、hashCode()、toString()等等）。

不过，通信协议是IIOP时，规则会稍有改变。通过narrow()操作，会向你提供一个可以强制转换的对象。

# PortableRemoteObject.narrow()

javax.rmi.PortableRemoteObject的narrow()方法会运行服务器开发商写的代码。不过，我们关心的只有一点，就是它要拿到JNDI返回的对象，并向我们返回一个真正实现home接口的对象。

换句话说，我们可以把它返回的对象强制转换为home接口类型，并调用create()。

```
PortableRemoteObject.narrow(o, AdviceHome.class);
```

从JNDI得到的对象

你希望这个对象就是这种接口类型，这必须是一个远程接口！

**Relax**

**你不需要了解IIOP的细节**

对于EJB，你不需要知道IIOP是怎么工作的，
除非你想开发非Java的CORBA客户。不过我
们不会这么做。起码在这里、这本书里还有考
试中都不会这样做。这些细节不在本书和考试
范围内。

但是…… 你确实需要知道一些，这就是规范要
求bean提供者有正确的假定，应该认为容器使
用的是IIOP，所以你的bean必须是IIOP兼容的。
但是谁把它变成IIOP兼容的呢？所以需要时就
要对远程桩进行窄化。还有另外几个IIOP小问
题，我们会在这一章里介绍。　关键是，你一
定要知道使用的是IIOP，不过除了这一点，其
他的不用知道太多。这一章我们会介绍有关
IIOP需要了解的所有内容。

**JNDI查找返回的home桩可能没有实现home接口！**

**你可能会得到一个不能强制转换为bean home接口的IIOP桩。这说明，你无法调用create()。**

**要得到一个可以强制转换为home接口的桩，必须首先对从JNDI查找（在bean home上的查找）得到的对象调用narrow()进行窄化。**

**（不过只有当home接口是远程home接口的情况下这才可行。）**



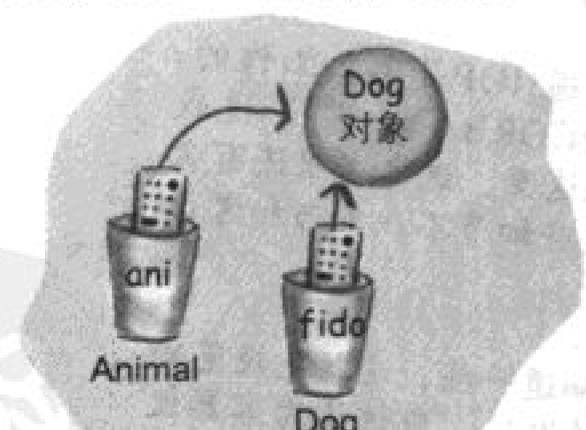

## 把窄化理解为一种“奇特的类型强制转换”

窄化和类型强制转换是不同的，不过，你可以把当想成一种“奇特的类型强制转换”。

类型强制转换与多态有关。利用类型强制转换，对象本身不变，但是你引用这个对象的方式会改变。利用窄化，你实际上会得到一个完全不同的对象！

**类型强制转换**

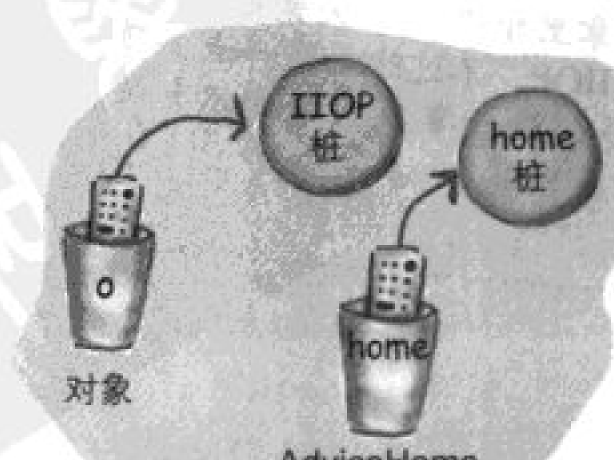
```
Animal ani = new Dog();
Dog fido = (Dog) ani;
```




类型强制转换允许你用多种方式来看一个对象。引用类型确定了你能调用哪些方法，但对象本身一直很清楚自己只是一个Dog。

**窄化**

```
narrow(o, AdviceHome.class);
```




窄化方法可能返回一个完全不同的对象（也可能不这样）。不过，不管怎样，你都会得到一个确实实现了接口的桩，这样就能对它进行类型强制转换。

## 既然我们（终于）拿到了home桩，下面用它来得到我们真正想要的东西吧……

**① 在home接口上调用create()，得到EJB对象桩**

```
Advice advisor = home.create();
```

这个创建方法返回组件接口（Advice）的一个引用。换句话说，它会返回EJB对象（实现了Advice，这是这个bean的远程组件接口）的一个桩。不需要对这个EJB对象桩进行类型强制转换和窄化。




**② 在组件接口（EJB对象桩）上调用一个业务方法**

```
System.out.println(advisor.getAdvice());
```

这里没什么特别的地方。这只是在Advice接口引用上的一个普通的方法调用。

不过，也不尽然。要记住，每个远程方法调用都声明了一个RemoteException！而且这是一个受查异常，所以你必须处理或声明这个异常。

```
try {
  System.out.println(advisor.getAdvice());
} catch(RemoteException rex) {
    rex.printStackTrace();
}
```

## there are no Dumb Questions

**问：行，我知道了，我不用去学IIOP的细节，但是我还是想了解一下为什么要使用IIOP。**

**答：** 好吧，既然你感兴趣，那我们就再多讲讲。IIOP是CORBA的通信协议，它比普通的RMI能表示更多的信息。例如，IIOP可以传播事务和安全信息……这些东西很重要，如果利用一个非IIOP远程方法调用是没办法传送这些信息的。

所以，有了IIOP，容器至少有了与其他服务器互操作的可能性，这甚至包括并非基于Java的服务器。

要记住，CORBA是一种标准（当然还不只如此），它能让用完全不同的语言编写的两个对象有可能调用对方的方法。

这并不是说你的服务器一定要使用IIOP。规范指出，你（开发人员）必须假设服务器在使用IIOP，这意味着你要保证你的bean是IIOP兼容的（本章稍后我们还会谈到IIOP兼容性）。

**问：如果我的服务器没有使用IIOP，我还要做窄化的工作吗？**

**答：** 可能需要，也可能不需要。如果只是做类型强制转换，你的代码没准也能正常工作。但是，（这个"但是"很重要），这样一来，你的代码就无法做到独立于开发商了！换句话说，如果没有使用窄化，你就会得到一个不可移植的应用，因为如果把这个bean重新部署在一个确实使用IIOP的服务器上时，就会导致客户出问题。

**问：使用窄化有什么不好的方面吗？特别是如果服务器没有使用IIOP的时候，窄化会不会有什么负面影响？**

**答：** 没有什么不好的影响（是这样，不管这里有多大的开销，和可移植性比起来，这点开销只能算微乎其微，做这么一点牺牲绝对值得）。如果你的服务器没有使用IIOP，那么narrow()就是一个不起任何作用的方法（也就是说，不做任何工作）。规范指出，窄化工作总是可以做的，即便真的不需要，它也不会对你造成什么危害。

**create()声明的返回类型是组件接口，而不是Object。**

**所以从create()返回的EJB对象已经知道自己是什么（组件接口的一个实现）。**

```
public Advice create()
```

home接口上create()方法的返回类型总是组件接口

所以，EJB对象桩不需要类型强制转换或窄化……

# 为会话bean编写远程home接口

既然已经从客户的角度了解了查找和创建过程，下面再来看为bean编写一个home接口时需要做些什么。对于会话bean，这个过程相当容易。实际上，对于无状态会话bean，简直是太简单了，你只要声明一个无参数的create()方法即可。

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;


public interface AdviceHome extends EJBHome {

   public Advice create() throws CreateException, RemoteException;

}
```

## home接口规则

1. 导入javax.ejb.* 和 java.rmi.RemoteException。
2. 扩展 EJBHome。
3. 声明一个create()方法，这个方法要返回组件接口，并声明一个CreateException 和一个RemoteException异常。

- 对于无状态会话bean，只能有一个 create()，而且它必须没有任何参数。
- 有状态会话bean可以有多个重载的 create()方法，而且不一定非要有一个无参数的 create()。
- 所有create()方法都必须声明一个CreateException 和一个 RemoteException异常，而且还可以声明其他应用（受查）异常。
- 有状态bean中创建方法的名字必须以“create”开头（createAccount()、createBigDog()、createFashionAdvisor()等等）。
- 对于有状态会话bean，参数必须是 RMI-IIOP兼容的（你应该已经知道了，这要求参数是Serializable对象、基本类型、远程对象，或者是Serializable对象、基本类型或远程对象组成的数组或集合）。

# 会话bean的远程home接口示例

这一页上都是合法的远程home接口的示例。你会看到，有一些可以同时是有状态和无状态的，而另外一些只能是有状态的（因为，这些接口有一个带参数的创建方法）。我们去掉了代码中的包（package）语句和导入（import）语句，这样好在有限的篇幅里放上更多重要内容。

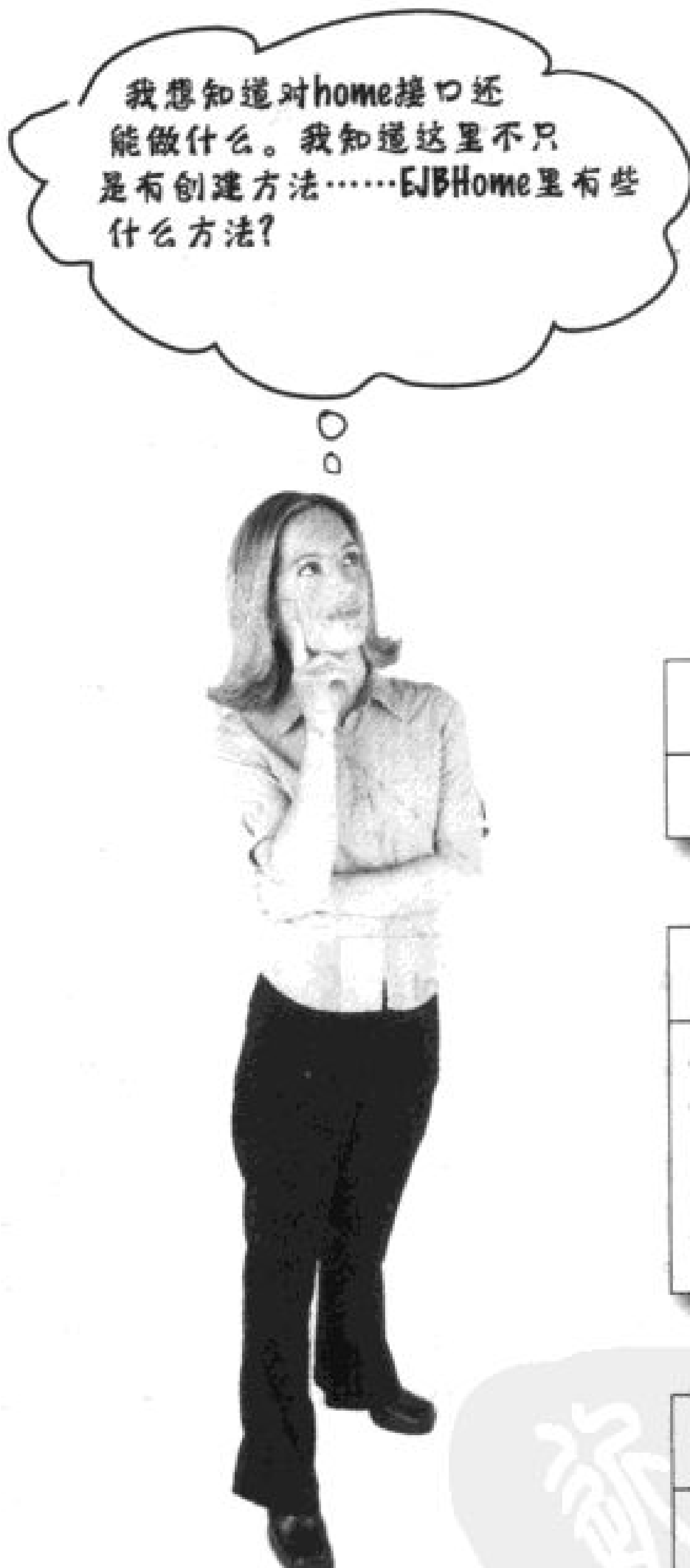
①
```
public interface CartHome extends EJBHome {
    public Cart create(String storeID) throws CreateException, RemoteException;
    public Cart create() throws CreateException, RemoteException;
}
```

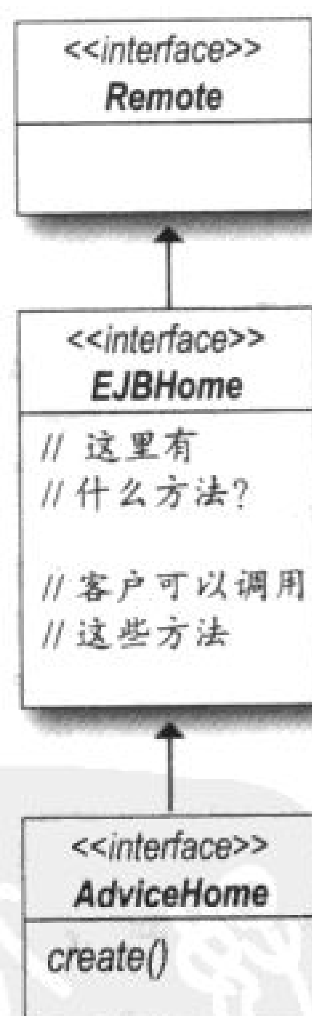
②
```
public interface MatcherHome extends EJBHome {
    public Matcher create(String customerID) throws CreateException, RemoteException;
    public Matcher createNewCustomer(String name, String login)
                                         throws CreateException, RemoteException;
}
```

③
```
public interface TicketsHome extends EJBHome {
    public Tickets create() throws CreateException, RemoteException;
}
```

④
```
public interface ClubHome extends EJBHome {
    public Club createExisting(String clubID) throws CreateException, RemoteException;
    public Club createNewClub(String clubName) throws CreateException, RemoteException;
}
```

这里只有一个接口可以作为无状态会话bean的home接口，也就是第3个。还可以注意到，第4个接口有两个创建方法，它们有同样的参数——String——但是这两个方法的名字不同，分别反映了这个特定的创建方法要创建什么。



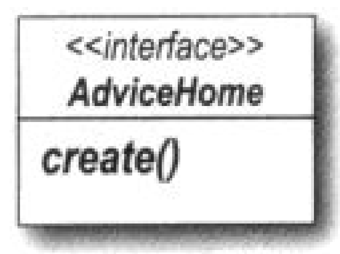




客户可以看到EJBHome里的所有方法！这些方法都通过AdviceHome接口展现给了客户。

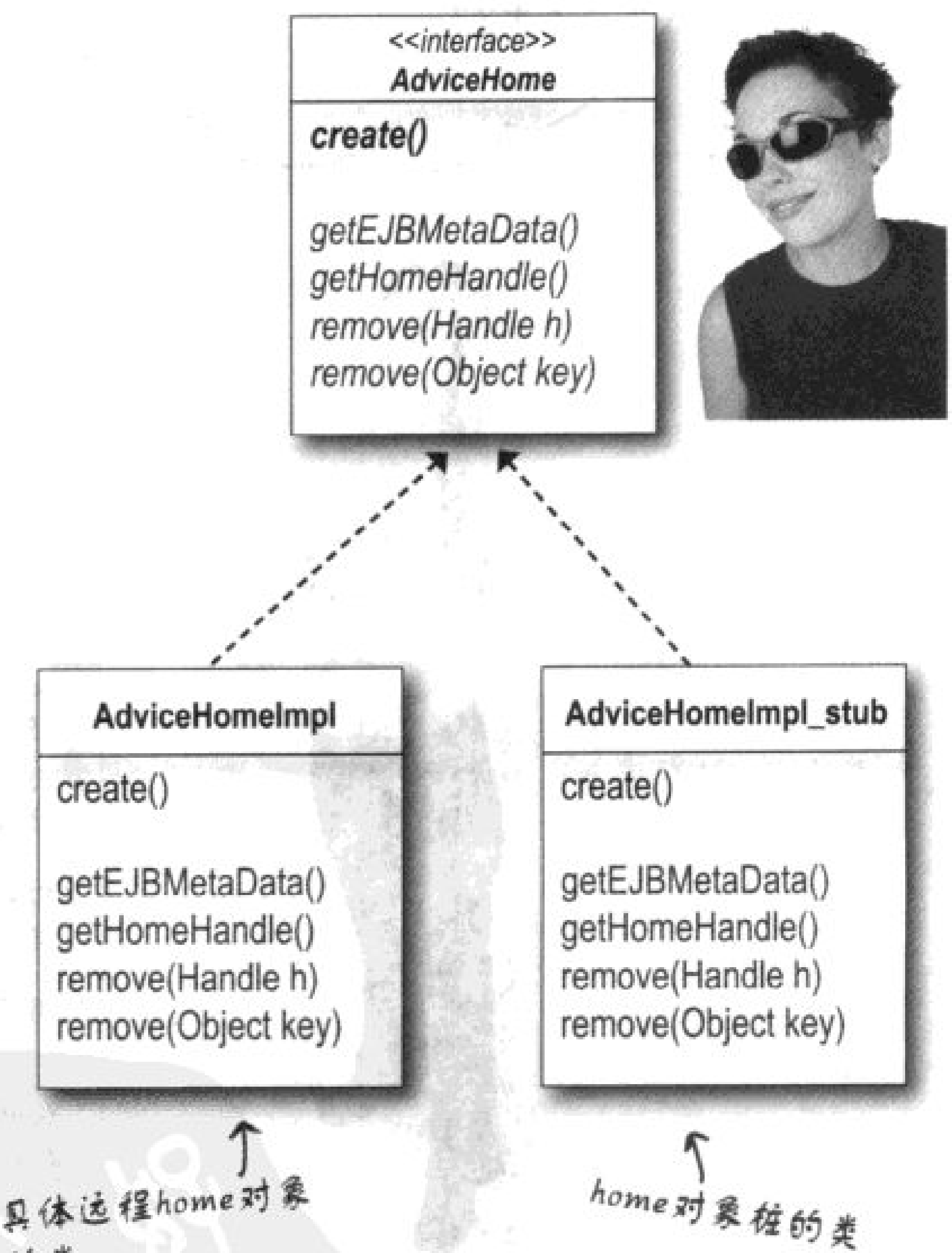
要记住，任何实现了AdviceHome的类不仅要实现AdviceHome的方法，还必须实现EJBHome的所有方法（若非Remote没有任何方法，否则连Remote的方法也要实现）。



## 你写的东西：




## 客户看到的东西：




**要记住，容器会实现你的home接口和与之对应的桩。**

**容器必须实现包括AdviceHome和EJBHome的所有一切。**

注意：这些不一定是实际的名字，这些类是服务器生成的，所以服务器可能会给这些类起任何名字。

## 如果你是一个客户，而且想……

## 可以调用这个方法：

**1 从bean得到反射之类的信息。**

除非你是一位工具开发商，否则一般不太可能调用这个方法。它会返回EJBMetaData接口，你可以用这个接口得到有关bean的更为特定的类信息。如果得到了一个EJBMetaData引用（通过调用getEJBMetaData()），就可以对它调用getHomeInterfaceClass()、getPrimaryKeyClass()、isSession()等等。

<<interface>>
EJBHome

**EJBMetaData getEJBMetaData()**
HomeHandle getHomeHandle()
void remove(Handle h)
void remove(Object key)

**2 对home串行化，这样以后就能再次使用这个home，而无需通过JNDI。**

假设你是一个客户，而且你在使用一个home，来建立一堆bean，在实体bean上调用home方法，等等。而且现在你必须重启机器，或者是要移到另一个机器上去。但是你还想继续使用这个home，该怎么办呢？

可以再回去找JNDI，通过它完成整个查找过程来得到这个home。不过，如果你向home要一个句柄，就能得到一个可串行化的对象，这样就能把它保存起来，以后再使用来得到home桩，而无需通过JNDI（稍后会对句柄做更多说明）。

<<interface>>
EJBHome

EJBMetaData getEJBMetaData()
**HomeHandle getHomeHandle()**
void remove(Handle h)
void remove(Object key)

**3 告诉home，你已经用完了一个会话bean。**

用完一个会话bean时，可以调用remove()，并传递EJB对象的句柄，把这个情况告诉home。没有错，确实是EJB对象。home对象可以给你一个句柄（这样以后你就能再取回home桩，而不用通过JNDI），与此类似，EJB对象可以给你一个它自己的句柄。（这一章我们还会介绍更多有关的内容。）
还可以对实体bean使用这个remove()方法，不过，对于实体bean，通常调用另一个版本的remove()方法更加容易。

<<interface>>
EJBHome

EJBMetaData getEJBMetaData()
HomeHandle getHomeHandle()
**void remove(Handle h)**
void remove(Object key)

**4 告诉home删除一个实体bean。**

注意，我们没有说“告诉home我们已经用完了一个实体bean”。这是因为，在实体bean上调用删除方法与在会话bean上调用这个方法截然不同。有关的细节将在“实体bean”一章中再做解释，不过可以这样简单理解：在删除一个实体bean时，并不只是告诉容器你用完了这个实体bean，而是告诉它所有人都已经用完这个bean。永远不会再用了。因为对一个实体bean调用remove()意味着：“从持久存储库里把这个实体永久删除吧。”（这通常可以理解为“从数据库里删除这一行”。）
这个版本的删除方法要取一个主键，会话bean是没有主键的，所以不同于上一个版本的remove()，这个版本只能用于实体bean。

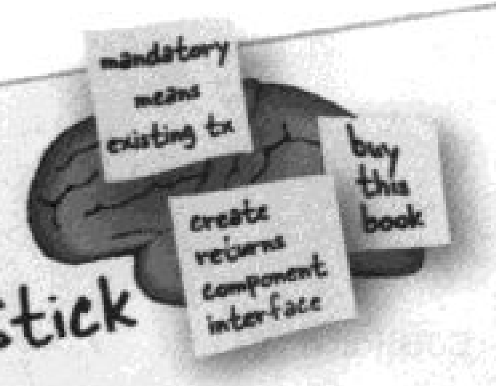
<<interface>>
EJBHome

EJBMetaData getEJBMetaData()
HomeHandle getHomeHandle()
void remove(Handle h)
**void remove(Object key)**



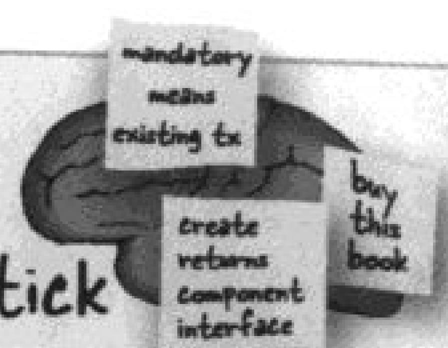

**问：** 这是不是说，客户必须知道他在使用一个会话bean，而不是一个实体bean？这不应该要求客户知道吧？

**答：** 不是这样的，客户得到一个会话bean的home接口时，他必须知道这一点，因为他不能调用remove(Object primaryKey)方法。如果他真的调用了，就会得到一个异常（javax.ejb.RemoveException）。这看上去好像是一个实现细节（也就是，这是一个会话bean还是一个实体bean），不应该要求客户知道的，但是在实际中，如果不知道在与一个会话bean还是实体bean通信，你就别指望能写出一个EJB客户。起码的，客户与实体bean home交互的方式与客户使用会话bean home的方式就完全不同。在关于“实体bean”的一章你会了解到它们之间存在的这种显著差别。




**Make it Stick**

客户上有一个home桩，
虚拟机对它很挑剔，
为了完成类型强制转换，
需要先做一个窄化，
因为它是CORBA兼容的。




**Make it Stick**

注意了，
不能从JNDI得到一个bean。

在JNDI上完成一个查找时，
只会得到客户要用的一个home。

## 要点

- bean的方法通过组件接口展现给客户。
- 客户不能直接得到bean的一个引用；客户必须通过bean的EJB对象，这个EJB对象实现了组件接口。
- 客户从bean的home得到bean EJB对象的一个引用。
- 要得到bean的home，客户要完成一个JNDI查找，这里要使用部署bean时所用的逻辑名。
- 要完成JNDI查找，客户必须首先得到一个InitialContext，这是进入服务器JNDI“虚目录树”的入口点。
- 对于一个远程home接口，从JNDI返回的桩必须得到类型强制转换和窄化。
- 窄化是一种“奇特的类型强制转换”，如果方法没有返回桩的客户接口，从这个方法返回的桩对象就要进行窄化。由于JNDI查找返回的类型是Object，所以查找返回的对象必须先窄化，然后再强制转换为bean的home接口。
- 对IIOP桩，窄化是必要的（IIOP是CORBA的通信协议），因为查找返回的对象可能不能实现多个接口，因此只知道类型Object中的方法。窄化可以返回一个实现了home接口的对象。
- home接口扩展了EJBHome，这个接口有4个方法（客户可以看到这些方法）：getEJBMetaData、getHomeHandle、remove(Handle h)和remove(Object primaryKey)。不能对会话bean调用remove(Object primaryKey)。

## Sharpen your pencil

根据会话bean home接口的规则，关于下面这个接口以下说法正确的是：

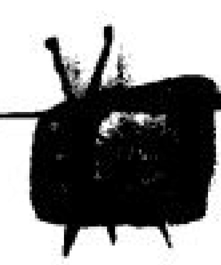
```
import javax.ejb.EJBHome;
import java.rmi.RemoteException;

public interface CartHome extends EJBHome {

   public Cart create() throws CreateException, RemoteException;

}
```

- ❑ CartHome肯定不是一个有状态会话bean的home。
- ❑ 这个接口少了一条import语句。
- ❑ 创建（create）方法少了一个异常。
- ❑ Cart肯定是这个bean的类类型。
- ❑ Cart肯定是扩展了EJBObject的接口。
- ❑ create()返回的对象必须窄化。
- ❑ create()返回的对象不需要类型强制转换。



**你必须会看代码，而且能推断些东西出来。**

考试时会让你看客户代码、接口代码或bean代码，从中推断出一些不能直接看到的东西。你必须了解home和组件接口的所有规则。下面就是组件接口的规则……

# home的内容就这么多了…… 下面来说说EJB对象，也就是组件接口，这是你真正想要的东西。

请记住，得到InitialContext、通过JNDI进行查找、类型强制转换、窄化还有创建，所有这些都只是为了得到你真正想要的东西，也就是有业务方法的某个对象。你可以用它得到bean，让bean做它该做的事情，比如数字处理、网上购物、提出建议等等。

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;


public interface Advice extends EJBObject {

   public String getAdvice() throws RemoteException;

}
```

## 组件接口的规则

1. 导入javax.ejb.* 和java.rmi.RemoteException。
2. 扩展EJBObject。
3. 声明一个或多个业务方法，这些方法都要抛出一个 RemoteException异常。
   - 参数和返回类型必须是 RMI-IIOP 兼容的（Serializable对象、基本类型、Remote对象，或者由上述类型组成的数组或集合）。
   - 可以有重载方法。
   - 每个方法都必须声明一个RemoteException异常。
   - 可以声明你自己的应用异常，但是这些异常不能是运行时异常（换句话说，它们必须是编译器检查的异常，即Exception的子类，而非RuntimeException的子类）。


我想知道对组件接口还能做些什么。我知道，它不只是有业务方法……EJBObject里有些什么方法？


<<interface>>
Remote
<<interface>>
EJBObject
// 这里有
// 什么方法？
// 客户可以调用
// 这些方法
<<interface>>
Advice
getAdvice()
与EJBHome接口一样，客户能看到EJBObject里的所有方法！这些方法都通过bean的组件接口（Advice）展现给客户。
任何实现Advice的类不光要实现Advice的方法，还必须实现EJBObject的所有方法。

# 想一下对你的EJB对象引用还想做些什么……

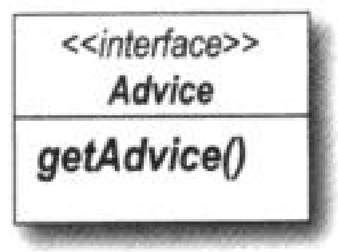
你是一个客户，你有AdviceBean组件接口的一个引用，而且你知道可以调用getAdvice()。但是，既然你已经那么费劲地拿到桩，你还想做别的什么事情吗？



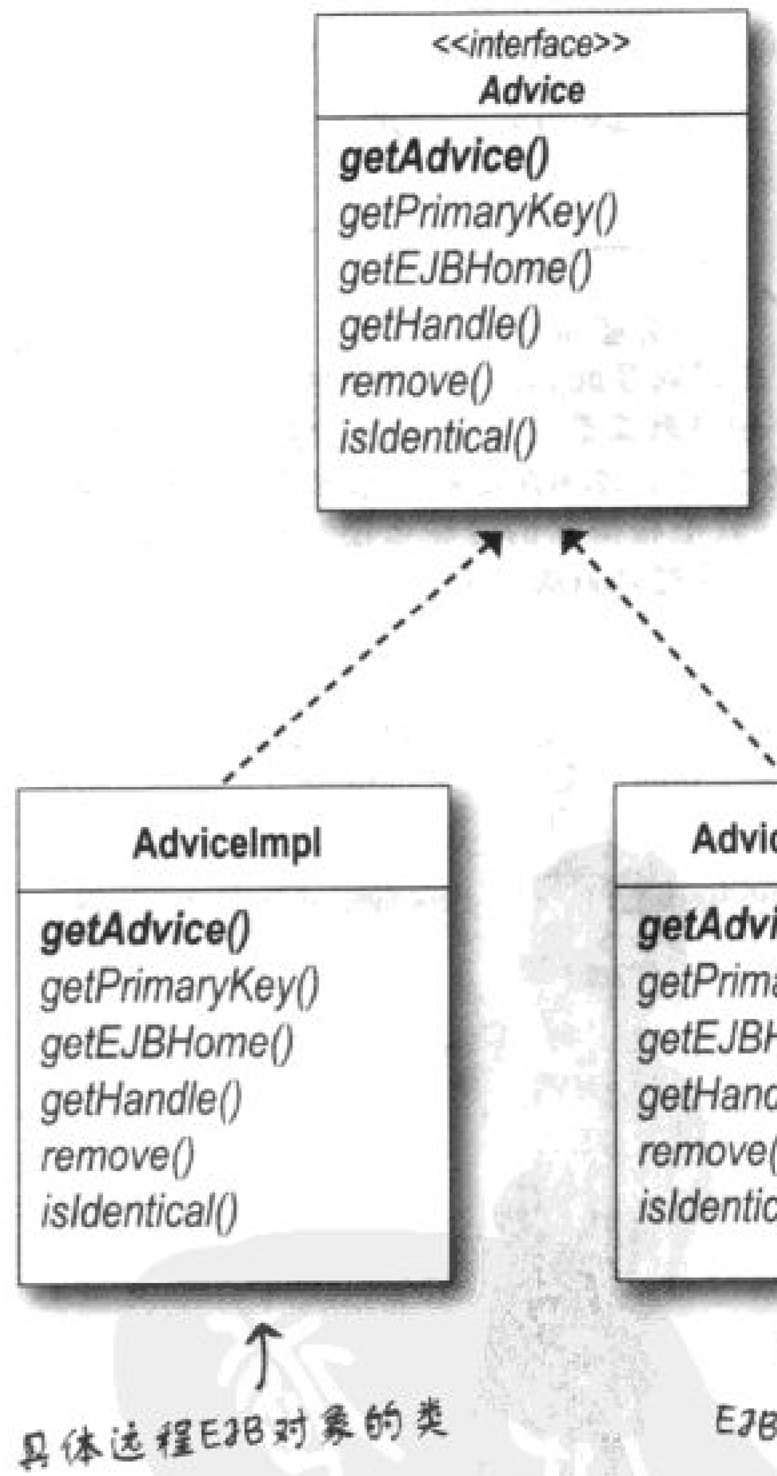


EJBObject（bean的保镖，提供业务方法）

## 你写的东西：




## 客户看到的东西：






要记住，容器会实现你的组件接口和与之对应的桩。

容器必须实现包括 Advice 和 EJBObject的所有一切。



**你最好深入地掌握这些方法。**

考试要求你知道EJBObject里的全部方法（共5个方法）。你必须知道，远程客户可以得到这些方法，而且要知道这些方法是怎么用的！

## 如果你是一个客户，而且想……

## 可以调用这个方法：

### 得到实体bean的主键

对这个内容我们现在先不做深入，因为只有对实体bean花很多笔墨才能把它讲清楚。只要知道这个方法不能用于会话bean就行了，因为会话bean没有一个惟一的身份要展现给客户。不过，如果客户以某种方式得到了一个实体bean的引用（可能是通过搜索顾客的名字来得到的），而且想要具体的主键，那就该调用这个方法。试着对一个会话bean调用这个方法看看，你会得到一个大块头的RemoteException异常（如果客户是本地的，就会得到一个EJBException异常）。

<<interface>>
**EJBObject**

**Object getPrimaryKey()**
*EJBHome getEJBHome()*
*Handle getHandle()*
*void remove()*
*boolean isIdentical(EJBObject o)*

### 得到bean的home

假设你得到了一个bean，但是没有bean的home。现在你想建立多个这种类型的bean。该怎么办呢？当然可以完成一个JNDI查找，用常规的办法得到home。但是，如果你没有足够的信息来完成JNDI查找，又该怎么办呢？你可以请求bean，让它把home的一个引用交给你。即使你能在home上完成JNDI查找，但相比之下，在bean上调用getEJBHome()会更加高效。

<<interface>>
**EJBObject**

*Object getPrimaryKey()*
**EJBHome getEJBHome()**
*Handle getHandle()*
*void remove()*
*boolean isIdentical(EJBObject o)*

### 保存EJBObject的一个引用

你在网上购物，精心挑选一些东西放到你的购物车里，你可能花了好几个小时“淘金”，为你的女朋友喜欢你穿浅蓝色还是军绿色举棋不定，好容易才下了决心。但是等你挑完了，却必须换到另外一台机器上。没关系。你可以向bean要EJB对象的一个句柄。可以用这个句柄得到你原来的EJB对象，再继续购物。

<<interface>>
**EJBObject**

*Object getPrimaryKey()*
*EJBHome getEJBHome()*
**Handle getHandle()**
*void remove()*
*boolean isIdentical(EJBObject o)*

### 告诉bean，你已经用完了

使用完这个bean后，要告诉它你已经用完了，这是一种礼貌，这样容器就可以释放这个bean为你保留的资源。注意危险！！这里只针对会话bean。尽管你也可以在一个实体bean上调用remove()，但是要记住，实体bean上的删除方法含义完全不同（这个内容将在关于实体bean的一章中介绍）。

<<interface>>
**EJBObject**

*Object getPrimaryKey()*
*EJBHome getEJBHome()*
*Handle getHandle()*
**void remove()**
*boolean isIdentical(EJBObject o)*

### 比较两个EJB对象，查看它们是否引用同一个bean

你得到了会话bean EJB对象的两个引用。现在你想知道，它们是不是确实引用了同一个bean。如果你在一个EJB对象上调用isIdentical()，而且isIdentical()方法取一个EJB对象引用作为参数，它会把这个作为参数的EJB对象与前一个EJB对象（isIdentical()就是在这个EJB对象上调用的）进行比较，返回true或false。

<<interface>>
**EJBObject**

*Object getPrimaryKey()*
*EJBHome getEJBHome()*
*Handle getHandle()*
*void remove()*
**boolean isIdentical(EJBObject o)**

## 网上购物要花时间，不要着急……

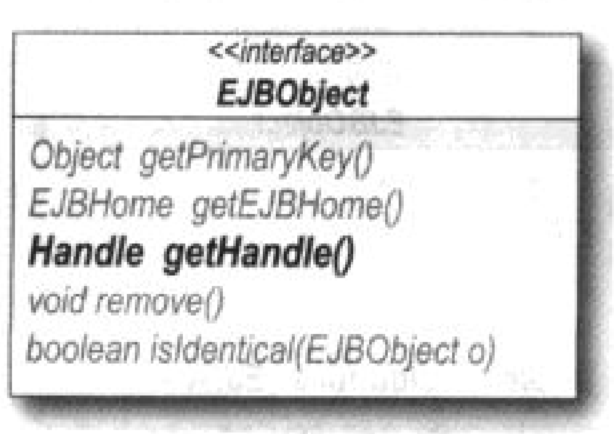
你在买东西。这可不是轻松的事情，因为你不能决定是买春天的还是买夏天的。你不想太着急，但是你已经在购物车里放了不少东西，而且发现上班已经迟到5分钟了。

你想一上班就继续买你的东西。

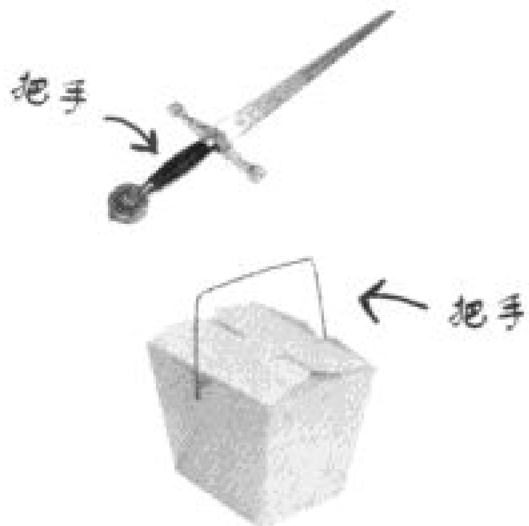
怎么做呢？如果你在Amazon上，那等你再登录时，你的购物车就会还在那里。不过，你现在用的是一个基于Swing的专用购物客户应用。你怎么把EJB对象从你家里的机器上拿到办公室的机器上呢？

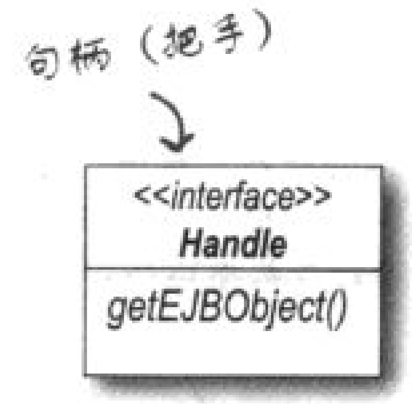
可以试试对桩串行化。不错，这样可能也行。不过再说一句，这样也可能不行。桩有一个“活动”的网络连接，你不能保证还能得到与同一个EJB对象的同一个连接。因为EJB对象是你自己的购物车bean的组件接口，这个bean是你个人的，而且是临时的，所以需要有一种办法能把同一个EJB对象再拿回来。

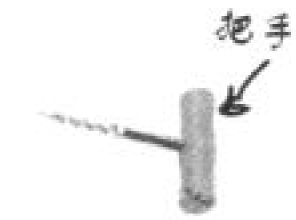


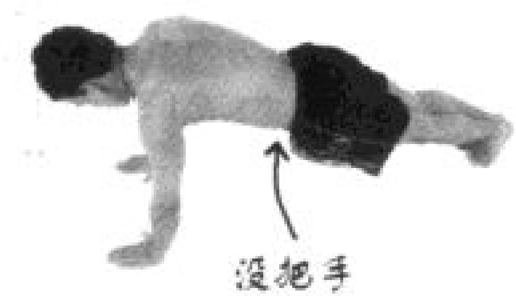



# 好在，我们拿到了句柄




句柄（handle，把手）可以解除你购物的后顾之忧。向bean（通过EJBObject接口）要一个句柄：

```
Handle myHandle = myCart.getHandle();
```

对它串行化，把它通过电子邮件寄给你自己，然后在你办公室的机器上逆串行化，这样就能拿到原来的EJB对象，可以继续购物了。

句柄是一个可串行化（serializable）的东西，它知道怎么来返回桩。它只有一个方法：

```
public EJBObject getEJBObject()
```

所以，当你调用这个方法时，必须对返回的桩进行类型强制转换和窄化！要记住，除非返回桩的方法所声明的返回类型是具体的远程接口，否则就要对它返回的桩进行类型强制转换和窄化。由于句柄的方法没有提示出你的组件接口是什么（例如，ShoppingCart），所以就像对待从JNDI lookup()方法得到的home桩一样，你面对的是同样的情况。需要对返回的桩进行类型强制转换和窄化。请注意，一定要做类型强制转换和窄化。

在你的客户代码中，可能有下面的一些代码：

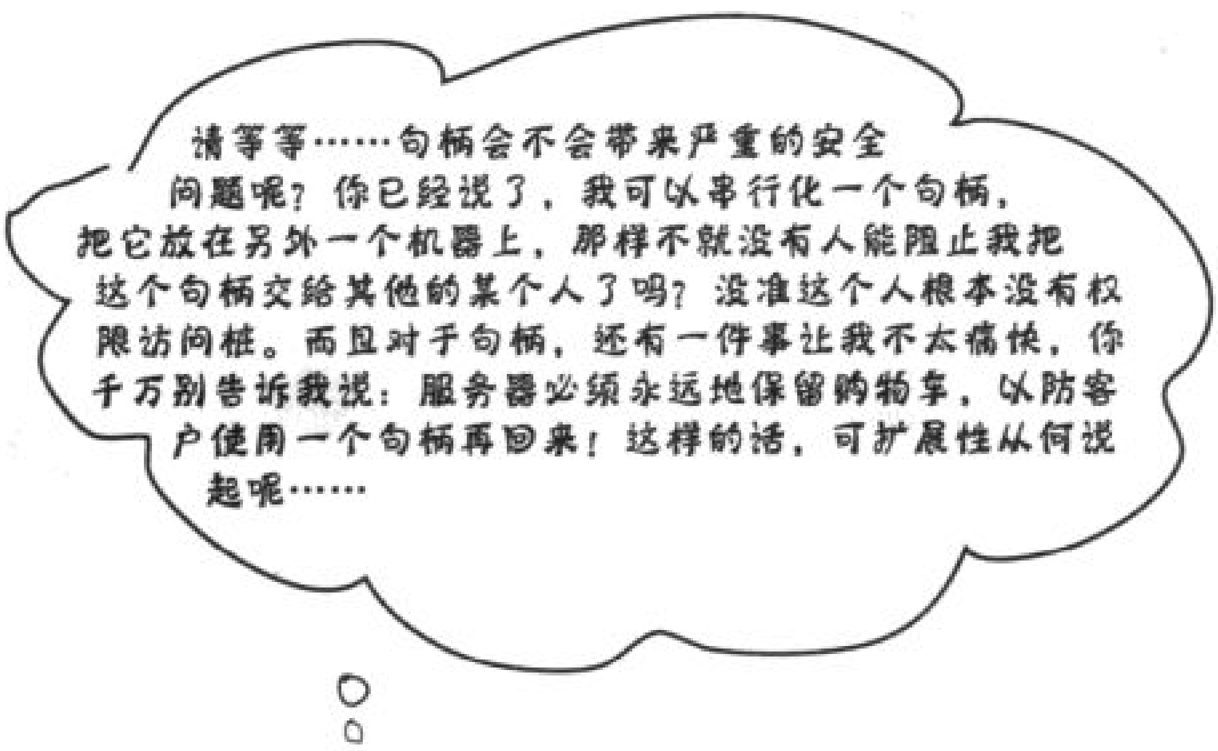
```
// 得到你以前保存的串行化句柄
Handle h = this.restoreTheHandle();

// 现在用它来得到EJBObject桩
Object o = h.getEJBObject();
Shopping cart = (Shopping) PortableRemoteObject.narrow(o, Shopping.class);
```

句柄是一个可串行化对象，它知道怎么拿回原来的远程EJB对象。句柄本身不是一个桩，但它能得到桩。

句柄只有一个方法，就是getEJBObject()，这个方法会返回一个类型为EJBObject的对象。

所以必须对得到的桩进行类型强制转换和窄化！






## 别担心！使用句柄不会破坏bean的安全性。

安全性建立在方法基础上，所以即使你把一个句柄交给别人，如果客户没有授权让他在这个bean上调用方法，那他从句柄得到的桩就没有任何用处。

## 还留有一个句柄并不意味着服务器仍然有你的bean。

如果你在购物，而且得到了一个句柄，然后服务器检测出你有一会儿没碰你的购物车了，那么服务器就会临时地把你的bean保存在一个存储资源中（这个过程称为钝化），但是你的购物车还是保留着，以备你再回来。但是，如果你在某个时间段里还是没回来，服务器就会觉得没希望了，而会把购物车删掉。这个bean就此成为历史。

在这种情况下，在句柄上调用getEJBObject()时，你的购物车并不在那里了，所以会得到一个RemoteException异常。

# isIdentical?

## 怎么看两个桩是否引用了同一个bean?



如果不看状态（无状态），双胞胎是完全一样的，因为她们都能做同样的事情，而且客户不会知道有什么区别。但是，如果双胞胎是有状态的，那她们是截然不同的。有状态的双胞胎并不一样，因为她们保留了特定于其各自客户的信息。

<<interface>>
**EJBObject**

*Object getPrimaryKey()*
*EJBHome getEJBHome()*
*Handle getHandle()*
*void remove()*
***boolean isIdentical(EJBObject o)***

如果你得到了两个桩，而且想知道它们是否引用了同一个bean，可以对一个引用调用isIdentical，你想让这个引用和谁比较呢？可以把你想与之比较的引用作为参数传递给isIdentical。这与使用equals()方法的方式是一样的。

难点在于，对于isIdentical()什么时候返回true，无状态会话bean、有状态会话bean和实体bean分别有不同的规则。

### 无状态会话bean

如果两个引用来自同一个home，即使这两个桩引用了两个不同的远程EJB对象，isIdentical()也返回true。对于服务器来说，来自同一个home的任何无状态会话bean都没什么不同，一个无状态会话bean与来自同一个home的其他bean（无状态会话bean）都一样，原因是客户无法区别它们有什么差别（因为这些bean无法保留任何客户特定的状态）。

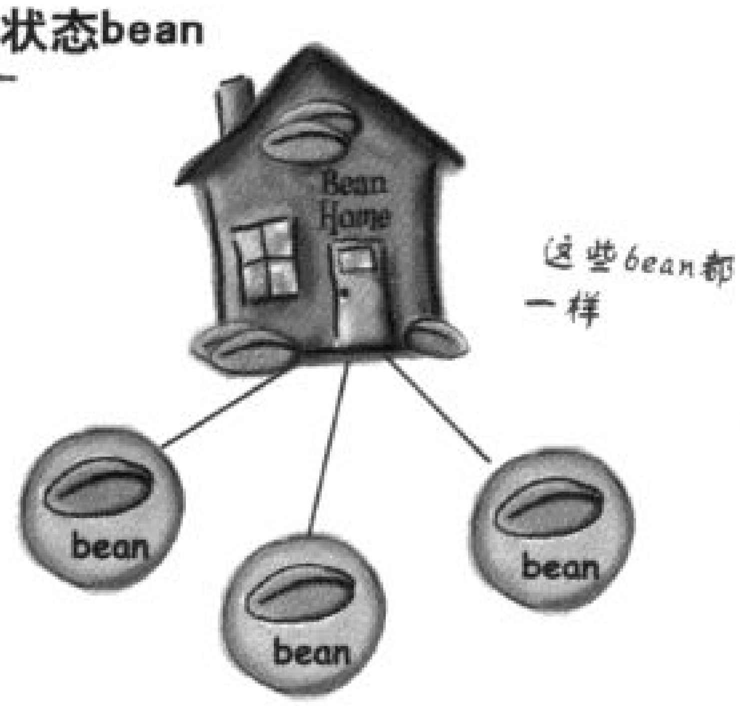
### 有状态会话bean

只要是两个不同的桩，这个方法肯定返回false，甚至来自同一个home的桩也不例外。毕竟，我的购物车是我的，你的购物车是你的，这是不一样的！

### 实体bean

如果两个桩引用了有相同主键的两个实体，这个方法就返回true。

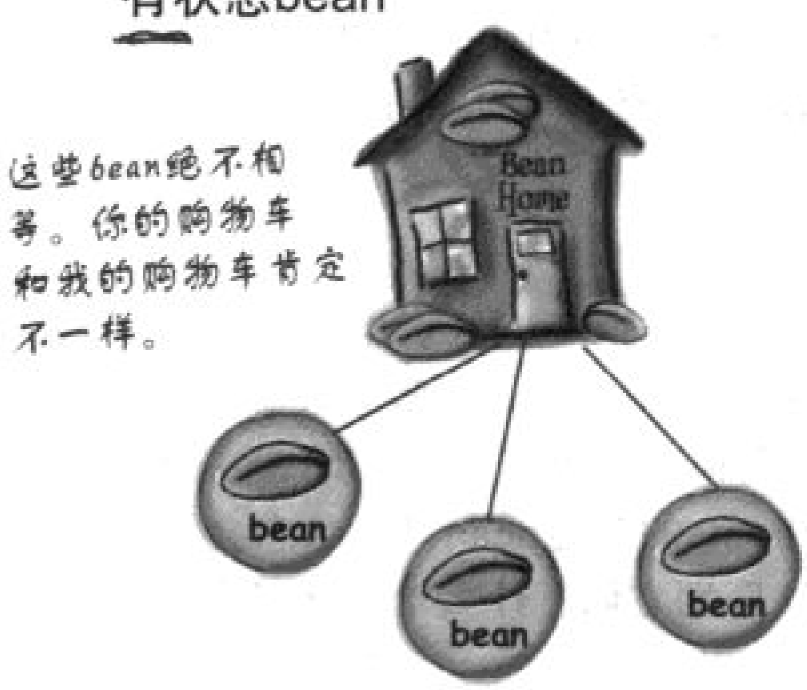
**无状态bean**




isIdentical()总是返回*true*，甚至是两个不同的bean。

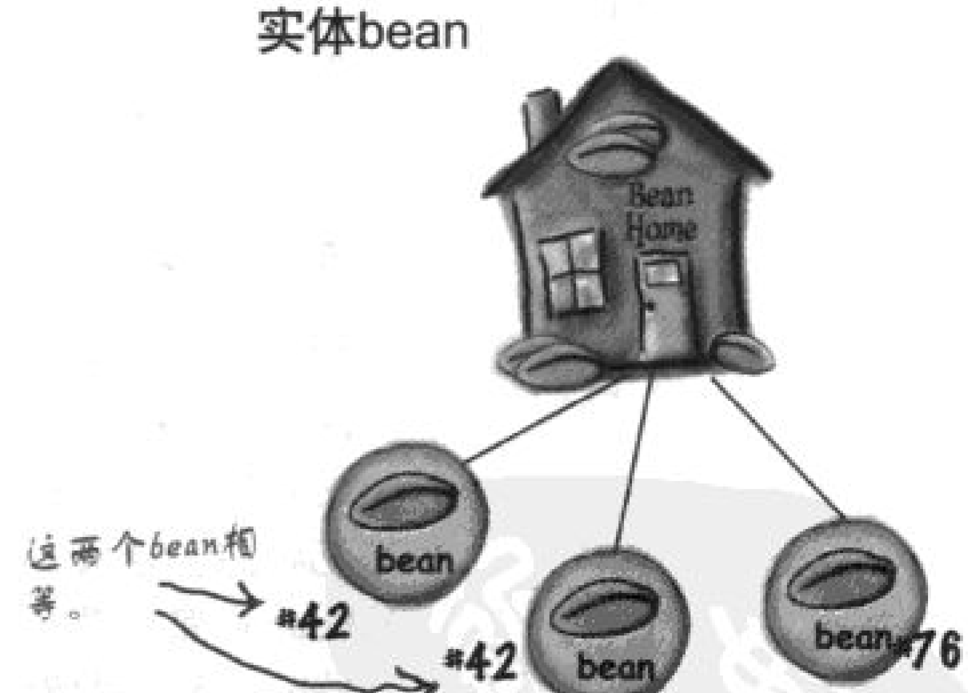
**isIdentical()方法就像是调用一个远程equals()方法……只不过你问的不是堆上的两个对象是否真正等价，而是服务器上的两个远程对象是否真正等价！**

**有状态bean**




isIdentical()对于不同的bean总是返回*false*，即使这些bean来自同一个home也不例外。

**实体bean**




对于引用同一个实体（换句话说，就是同样的主键）的bean*，isIdentical()返回*true*。

*这里我们用“bean”不太严格，因为从概念上讲，服务器只使用一个bean表示一个特定实体。所以主键为#42的bean只有一个，但是客户对它可能有多个EJB对象引用。

## there are no Dumb Questions

**问：为什么不能用equals()方法而要用isIdentical()呢？equals()不就是做这个的吗？**

**答：** 记住，我们讨论的是远程对象。equals()方法比较的是同一个堆（HEAP）上的两个对象，而isIdentical()比较的是服务器上的两个远程对象。

**问：我还是不清楚为什么不能在桩上实现equals()方法来做同样的事情。**

**答：** 首先,equals()方法不是一个远程方法。在桩上总是可以调用equals()，因为可以在堆中的任何对象上调用这个方法。但是它不是远程接口的一部分，所以这不是一个远程方法（例如，它没有声明RemoteException异常，不满足远程方法的一些特征）。

要记住，使用equals()方法是要看堆中的两个对象是否真正等价。开发商可以按自己喜欢的方式来实现比较桩的equals()方法，但是这对于服务器后台上所发生的并不能说明什么。如果两个桩没有通过equals()测试（也就是调用equals()返回false），这并不表示服务器也这样认为，也就是说，服务器仍有可能认为这两个EJB对象相等（或引用了相等的bean）。

例如，你的服务器可能在使用RMI桩，RMI桩的比较与两个EJB对象的实际比较之间有什么关系，这在RMI桩中没有任何规定（没有这方面的逻辑）。RMI桩了解远程对象，但不知道这些远程对象表示的是什么。

**问：EJBObject接口中怎么有一个得到bean home的方法呢？如果没有home，怎么得到原来的bean呢？？？**

**答：** 要得到一个EJB对象的引用，还有其他一些方法。使用JNDI来查找EJB对象肯定是不行的；因为注册的只是home。

不过……把EJB对象引用作为参数或返回值传递则是允许的。在一个bean中可能有这样一个业务方法，它惟一的任务就是给你传回另一个bean的EJB对象引用。

下面假设得到了这样一个EJB对象引用，你并没有这个EJB对象相应bean的home，下面你想得到更多这种bean。此时就可以使用getEJBHome()来请求得到bean的home。

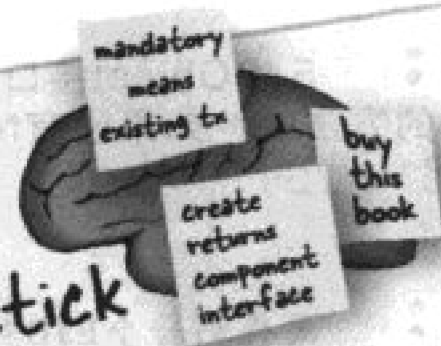
即使你确实有足够的信息完成一个JNDI查找来查找bean的home，相比之下，JNDI查找代价会昂贵得多。如果只是从bean直接得到home引用，就能节省一些开销。

**问：为什么不能对桩串行化？为什么需要句柄？**

**答：** 我们没有说桩不可串行化。但是，即使对桩串行化，也不说明这就能得到足够的信息再拿到原来那个（或者意义上相等的）EJB对象。从服务器得到桩时，它知道怎么和一个特定的远程对象联系。重新创建这个桩时，那个特定的远程对象可能已经不复存在了。

**问：那么句柄能有什么作用？**

**答：** 句柄很聪明，它知道怎么和服务器通信，怎么交回一个与你原来那个EJB对象相同的EJB对象。换句话说，这可能不是同一个EJB对象，但是客户无法区别。




**Make it Stick**

在你的无状态的home里，
只写一个方法，不用多写，
客户还是能看到
另外的4个。
[方法]

注意了，
从EJBObject，客户能得到5个。
[方法]




**当心！** 在EJB中，“can”有不同的含义。

考试的时候，有的人会被“can”这个词搞糊涂，比如说“会话bean客户能（can）在EJB对象桩上调用getPrimaryKey()吗？”

这要看你怎么定义“can”了。如果是考试，上面这个问题的答案就是否定的！换句话说，因为已经向调用者合法地展现了方法，这说明调用者如果有这个方法调用，则是能够编译的，但是如果规范说不能，那实际意义上就不能这么做。所以，只是能（编译）并不意味着你能调用（按规范的意义来说）。要记住，这里不仅有一个编译器法则，还有一个EJB规范法则。考试时，我们问的是EJB法则。如果考试确实涉及到编译，你会从措词上区别出来。

## 要点

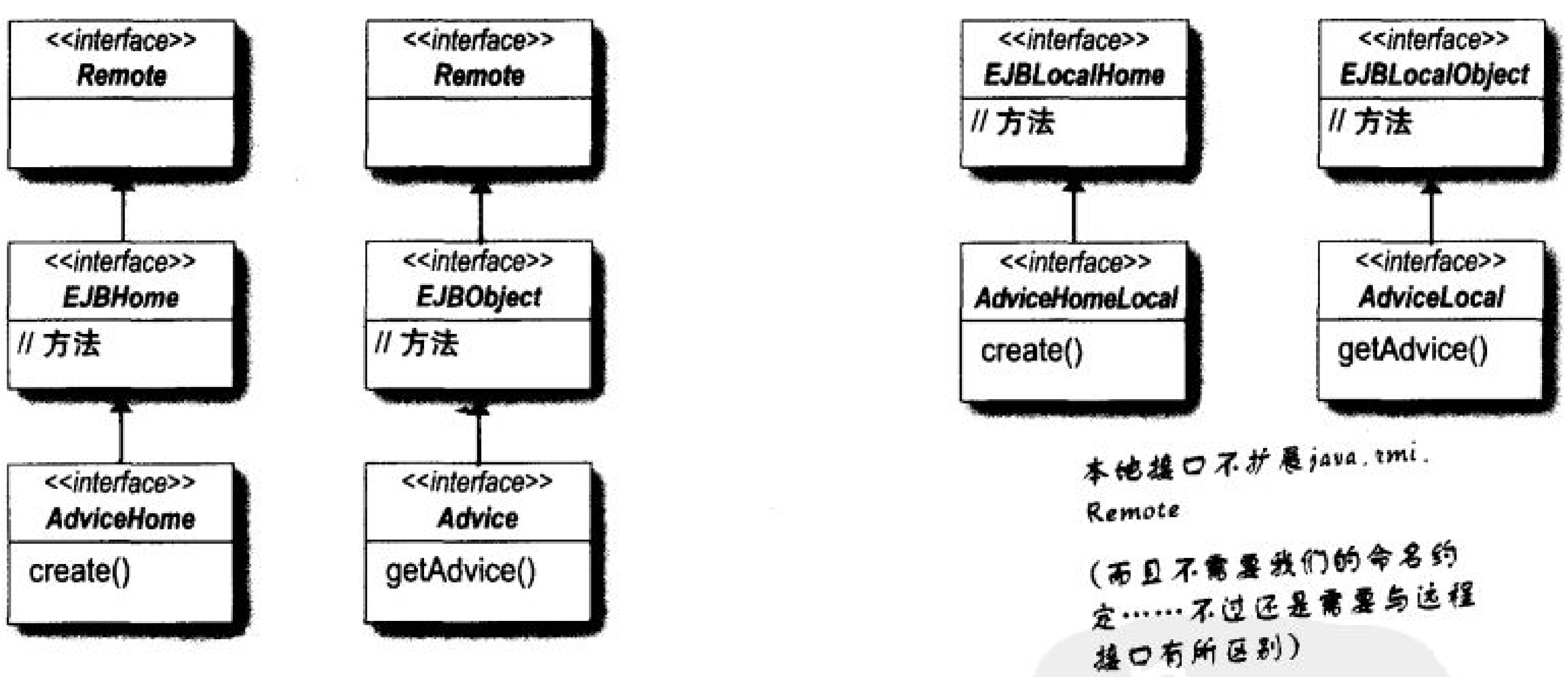
- 在组件接口中展现bean的业务方法。
- 远程组件接口必须扩展javax.ejb.EJBObject。
- 客户通过在bean的home接口上调用一个方法来得到bean的EJBObject引用。
- 有状态和无状态会话bean的引用都从home的create()方法获得。
- 从EJBObject接口，客户会看到另外5个方法：getEJBHome()、getHandle()、remove()、isIdentical()和 getPrimaryKey()。
- 只有实体bean客户允许在bean的组件接口上调用getPrimaryKey()。对于会话bean客户，调用这个方法会得到一个RemoteException异常。
- getEJBHome()方法返回bean home接口的一个引用，这样如果你想得到更多该类型的bean，客户就不必通过一个JNDI查找来得到home了。
- getHandle()方法返回一个Serializable对象，以后可以用这个对象来重新建立与服务器的联系，并返回组件接口的桩，客户就用这个桩来得到句柄。
- 句柄有一个方法，getEJBObject()，它会把远程桩作为EJBObject类型返回。这说明，这个桩就像从JNDI查找得到的home桩一样，必须得到类型强制转换和窄化。
- isIdentical()方法有些像在服务器上完成一个equals()方法。对于来自同一个home的两个不同的无状态bean，它会返回true，而对于来自同一个home的两个不同的有状态bean，isIdentical() 方法会返回false，对于有相同主键的实体的引用，则会返回true。

## bean的客户接口可以是本地的

到目前为止，我们只介绍了bean的远程客户接口，不过在EJB 2.0中，会话bean和实体bean还可以展现一个本地客户视图。换句话说，这是没有扩展java.rmi.Remote的客户接口。

这说明了什么？这说明home对象和EJB对象不是远程对象！它们与客户和bean在同一个JVM中运行。在关于实体bean容器托管关系（CMR）的一章中，我们将了解到为什么规范中要增加本地接口。不过就目前而言，把本地接口想成是一种非常特殊的情况就行了。

**远程客户视图** **本地客户视图**

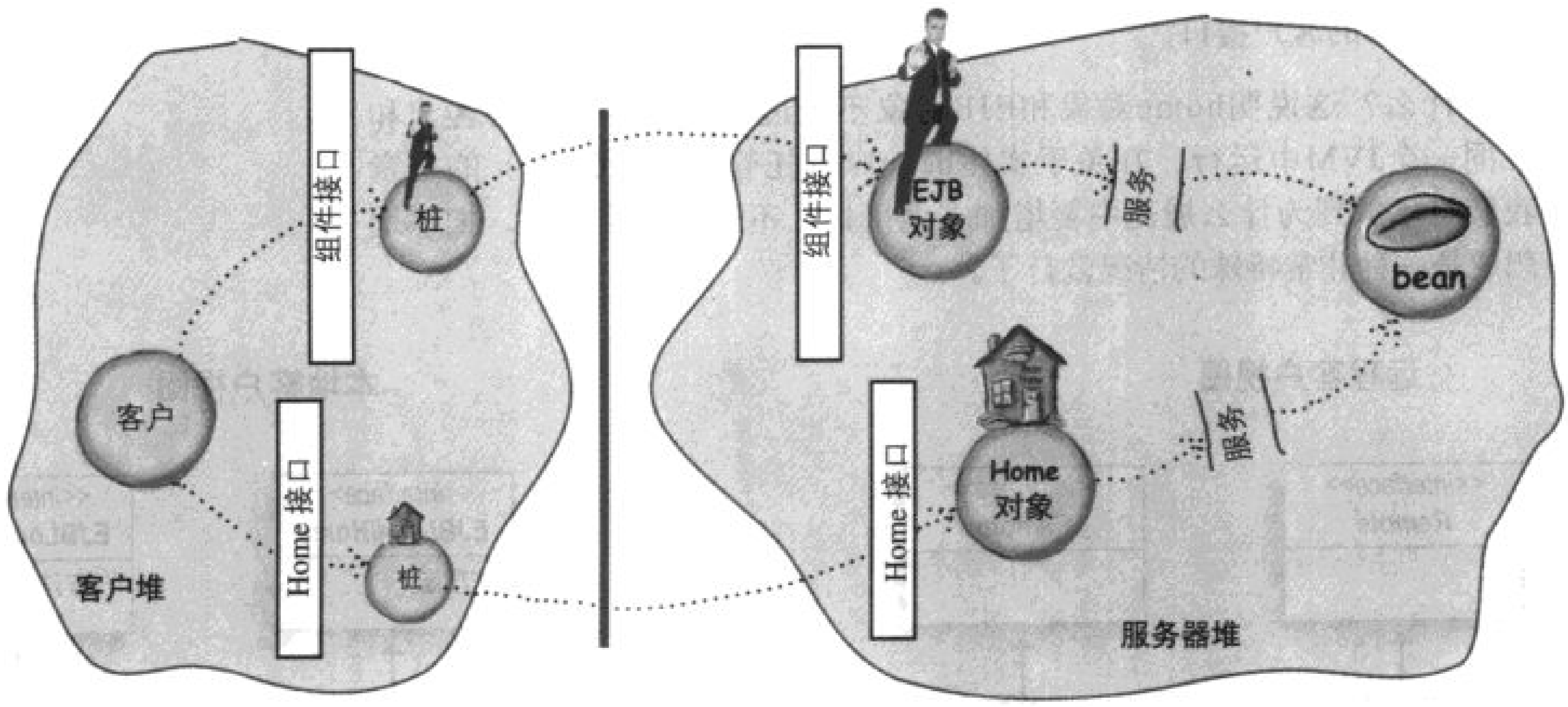



**BRAIN POWER**

和本地接口EJBLocalHome和EJBLocalObject相比，远程接口EJBHome和EJBObject中的方法更多一些。回顾本章，看一下EJBHome和EJBObject有哪些方法，得出这些远程接口里的哪些方法不适合或者不需要放在本地接口里。

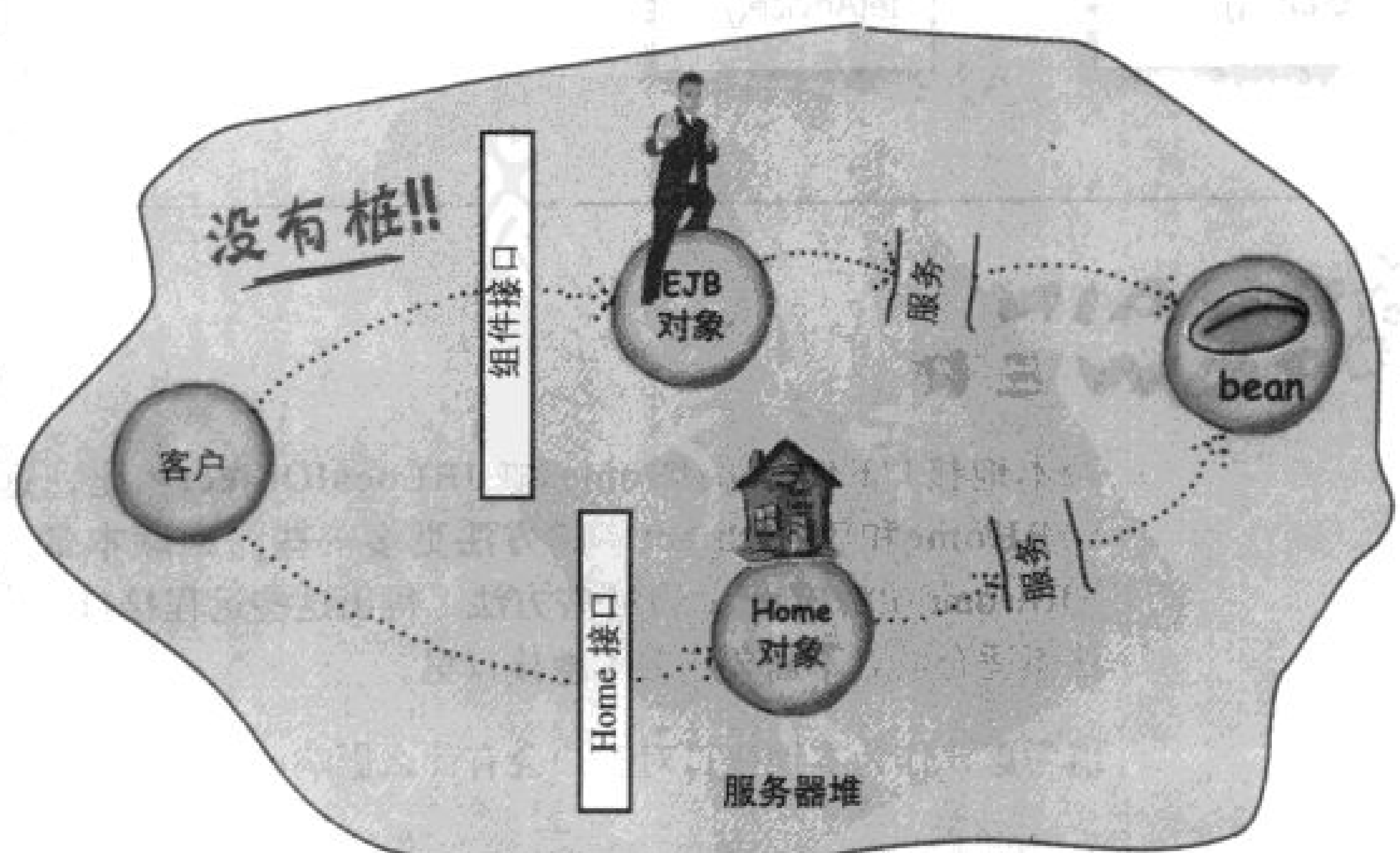
请考虑，作为本地接口对客户会有什么影响……

## 远程客户视图




## 本地客户视图

客户还是不能直接访问bean，因为服务器还需要一个地方来截获对bean的调用（这样服务器就能增加服务了）。但是对于本地客户，客户有home和组件接口对象的一个本地引用。



## Sharpen your pencil

**假设客户有home和组件接口（也就是说，home对象和EJB对象）的一个普通的Java引用，你觉得远程接口中的哪些方法可以放在本地客户视图中？**

换句话说，EJBHome中的哪些方法不在EJBLocalHome中，EJBObject的哪些方法在EJBLocalObject中没有？*为什么？*



<<interface>>
**EJBHome**

- *EJBMetaData getEJBMetaData()*
- *HomeHandle getHomeHandle()*
- *void remove(Handle h)*
- *void remove(Object key)*

EJBHome中有4个方法，EJBLocalHome只有其中的一个，是哪一个？

<<interface>>
**EJBLocalHome**

1 ______________________



<<interface>>
**EJBObject**

- *Object getPrimaryKey()*
- *EJBHome getEJBHome()*
- *Handle getHandle()*
- *void remove()*
- *boolean isIdentical(EJBObject o)*

EJBObject中有5个方法，而EJBLocalObject只有4个（而且这4个中的一个与远程接口中的版本还稍有区别）。这4个方法分别是什么？

<<interface>>
**EJBLocalObject**

1 ______________________

2 ______________________

3 ______________________

4 ______________________

## 对于本地客户接口，哪些方法有意义？

你已经知道了，相对于远程客户接口，本地客户接口中少了一些方法。下面来看哪些方法少了，为什么没有这些方法。

### 本地接口需要句柄吗？

还记得EJB中为什么会存在句柄吧，句柄是为了给你一个可串行化的对象，你可以使用这个对象重新建立你原来所用EJB对象的一个桩。句柄就是一个远程连接的抽象。所以……你觉得句柄在本地客户上还有意义吗？

### 本地接口需要EJBMetaData吗？

还记得EJBMetaData是干什么的吧，它用于得到有关一个bean的反射之类的信息。如果在一个bean的远程home上调用getEJBMetaData()，就会得到一个实现了EJBMetaData的对象。这个接口（EJBMetaData）有一些方法，允许你向bean讨教，对构成组件的类有更多了解。所以说，本地客户还需要EJBMetaData吗？

### 本地接口需要isIdentical()吗？

要记住为什么存在isIdentical()，这个方法允许你比较两个home或组件接口引用，查看它们是否引用服务器上“意义上等价的”两个bean。本地客户上需要使用isIdentical()吗？（重要提示：服务器可以采用自己选择的任何方式来实现 .equals()方法……）

### 本地接口需要主键信息吗？

还记得主键为什么存在吗？主键是为了惟一标识实体bean。你需要标识本地客户上的一个实体bean吗？

### 本地接口需要删除方法吗？

还记得remove()是干什么的吧，对于会话bean，这个方法是告诉容器你已经用完了bean，或者对于实体bean，这个方法要告诉容器永久地删除实体。本地客户需要在一个bean上调用remove()吗？

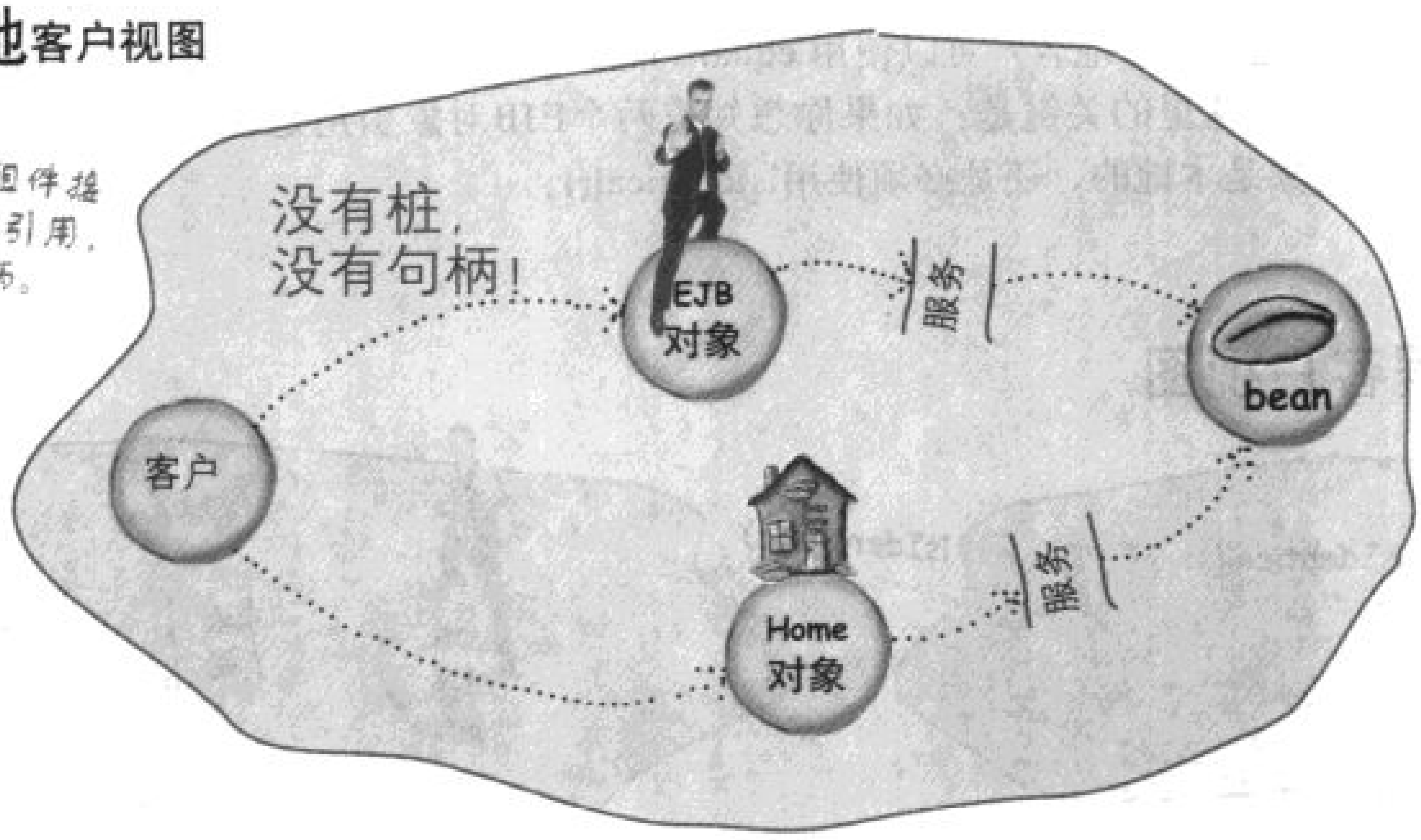
# 想到句柄就应该想到远程

## 本地客户不需要句柄！

本地客户没有用到句柄，因为句柄的目标很明确，就是得到一个可保存的（或者Serializable）对象，这个对象知道怎么重新建立与远程对象的通信。

### 本地客户视图

本地客户有home或组件接口的一个普通的Java引用，所以没有必要用句柄。




# 得到反射时谁需要EJBMetaData？

## 本地客户不需要EJBMetaData！

利用Java反射API，可以询问一个对象，得到有关这个对象的类的各种信息。对于远程对象，这是办不到的，因为无法得到远程对象的类的引用。惟一能向远程客户要的只是桩对象，但是桩对象无法告诉你有关实际EJB对象或Home对象的任何信息。

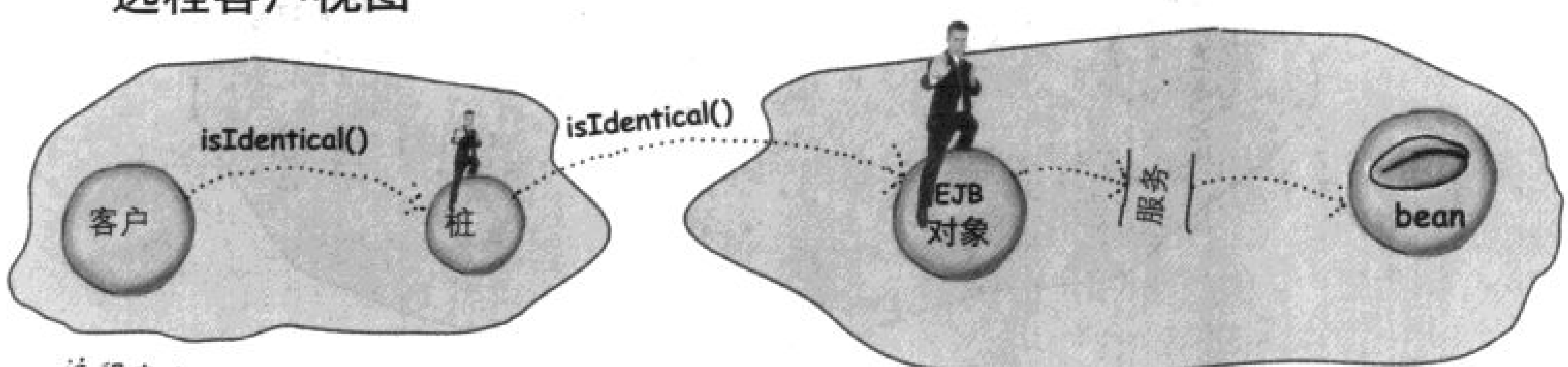
所以，远程home客户必须使用EJBMetaData（从EJBHome getEJBMetaData()方法返回的接口），才能得到有关的信息，而本地客户则不同，它只需使用Java反射方法（getClass()等等）就行了。

# 既然已经有了equals()，还需要isIdentical()吗？

## 本地客户还需要isIdentical()

对于一个远程客户，应该还记得，所做的惟一的本地比较是对两个桩对象的比较，即使用equals()。如果你想对服务器上的东西进行比较（在这里就是比较两个EJB对象引用），这个方法是不行的。而这正是isIdentical()要做的。不过本地客户有实实在在的引用！它们有EJB对象的实际引用，所以可以使用equals()来看两个EJB对象（本地）引用在意义上是否等价。不过……这并不是你想要的。规范中没有明确保证.equals()最后会得到什么结果！所以，尽管看上去对本地客户可以使用.equals()而不用isIdentical()，但是规范不能保证这二者的结果相同。这里的关键是：如果你想知道两个EJB对象引用是否引用同一个会话对象，即使EJB对象是本地的，还是必须使用isIdentical()。

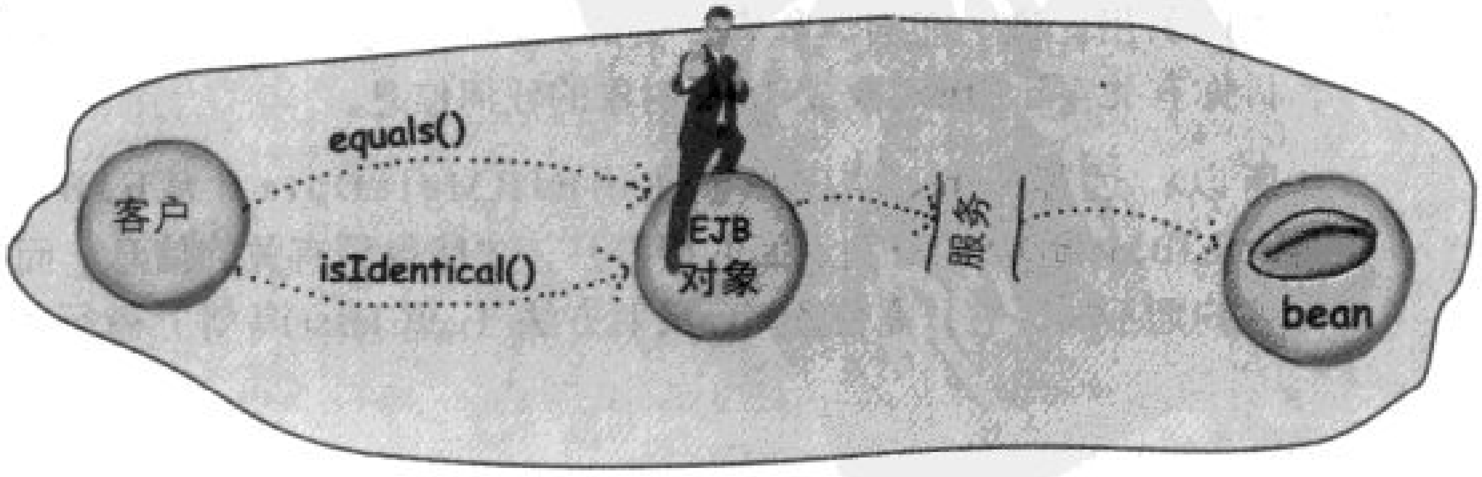
### 远程客户视图




远程客户必须做一个远程方法调用(isIdentical())，来得到一个EJB对象的有意义的比较信息。对桩调用equals()得不到任何有用的信息。

### 本地客户视图

在一个本地EJB对象引用上调用equals()和isIdentical()的结果不能保证一定相同。还是需要isIdentical()来确定两个EJB对象引用是否引用同一个会话对象。



# 为什么这么多删除方法？

## 对于远程客户，home上有两个删除方法，组件接口中有一个删除方法

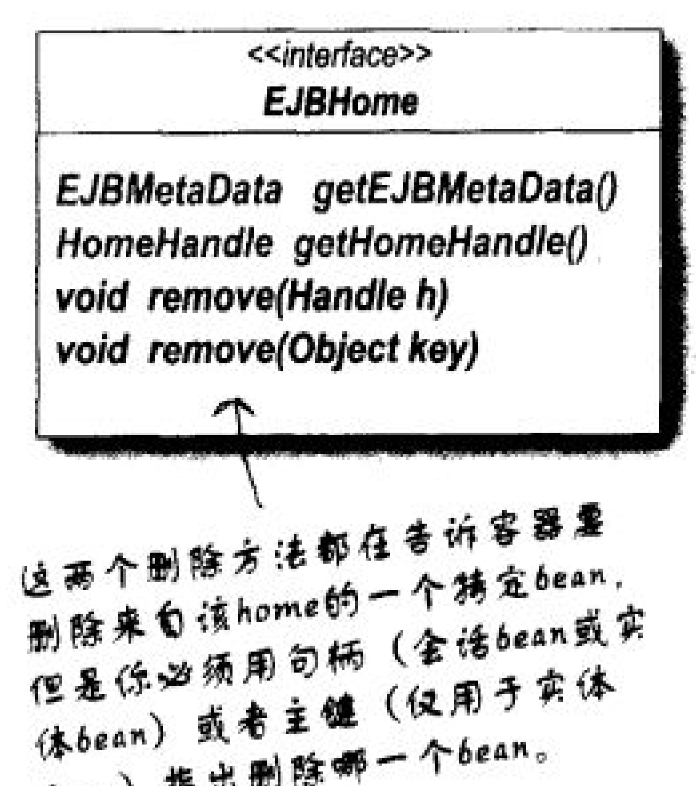
要记住，远程客户有三个可用的remove()方法，其中两个在home接口中，还有一个在组件接口中。EJBObject中的remove()很简单，如果调用这个方法，就是说你想删除相应的那个bean！换句话说，你在哪个bean的EJBObject上调用remove()，就要删除哪个bean。对于会话bean，要记住，调用remove()只是告诉容器你已经用完了这个bean。这是一种很好的做法，这样可以改善可扩展性，因为服务器不用再为你保留特定于客户的资源，而只是等着，一旦你的购物会话有一段时间没有活动，而且达到超时时限，就会把bean删除掉。

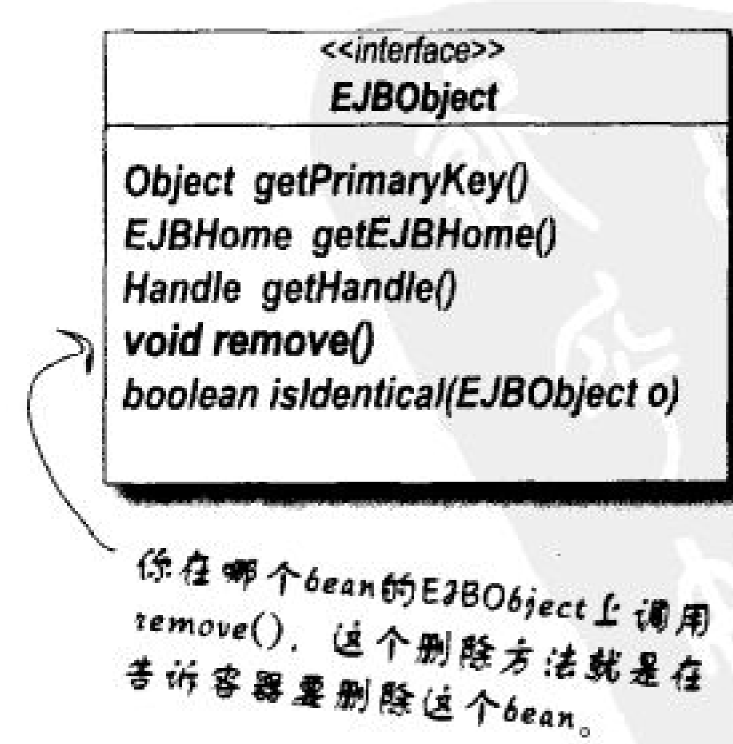
不过，要是调用home上的删除方法就没有这么简单了。首先，实际上你是无法删除一个home的！服务器会一直保留bean home，无论你有没有用它，所以这里没有太多特定于客户的资源。也没有必要告诉服务器你已经用完了home，因为服务器只会说："那又怎么样？"

那么调用home上的remove()是什么意思呢？

这说明你在告诉home删除某个来自该home的bean，而且这意味着你必须标识出要删的是哪个bean！

**远程客户视图**



## 为什么home里有一个无参数的删除方法……



**既然没有句柄，怎么能用一个取句柄为参数的删除方法呢？**

**本地客户没有句柄，所以本地home没有取句柄为参数的remove()。**

如果本地home接口没有句柄，那就不会有一个取句柄为参数的删除方法。这是因为，为了向home传递一个句柄（这个句柄惟一地标识了你要删除的那个bean），你首先要得到bean的句柄。但是由于本地化bean没有句柄……这下你就能看出问题了吧。

there are no Dumb Questions

**问：** **如果有一个本地home，而且又有一个远程EJB对象引用会怎么样呢？能不能把Remote bean的句柄传给这个本地home呢？**

**答：** 不行!! 因为不能把本地接口和远程接口混在一起。从本地home出来的只能是本地bean，反之亦然。所以若是有一个Remote bean的句柄，要把它提供给这个Remote bean的home，除非这个home也是远程（Remote）home才行。

上面说得好像不太清楚……好吧，这样来想想，远程home只会分发这个bean类型组件接口（EJBObject）的远程引用，而本地home只会返回组件接口（EJBLocalObject）的本地引用。

**问：** **再告诉我一次为什么不能删除home。**

**答：** 为什么要删除home（也就是说，告诉容器你已经用完了这个home），没理由这样做呀，因为不管你有没有兴趣，容器都会保留home。所以，如果你说要删掉home了，容器就会说："喂，别自以为是了。我在服务器上做的又不是完全为你一个人。你要是用完了你的home引用，我不管不就得了。"

**在home上调用删除方法不是要删除这个home。**

**在home上调用删除方法只是告诉home删除一个bean。**

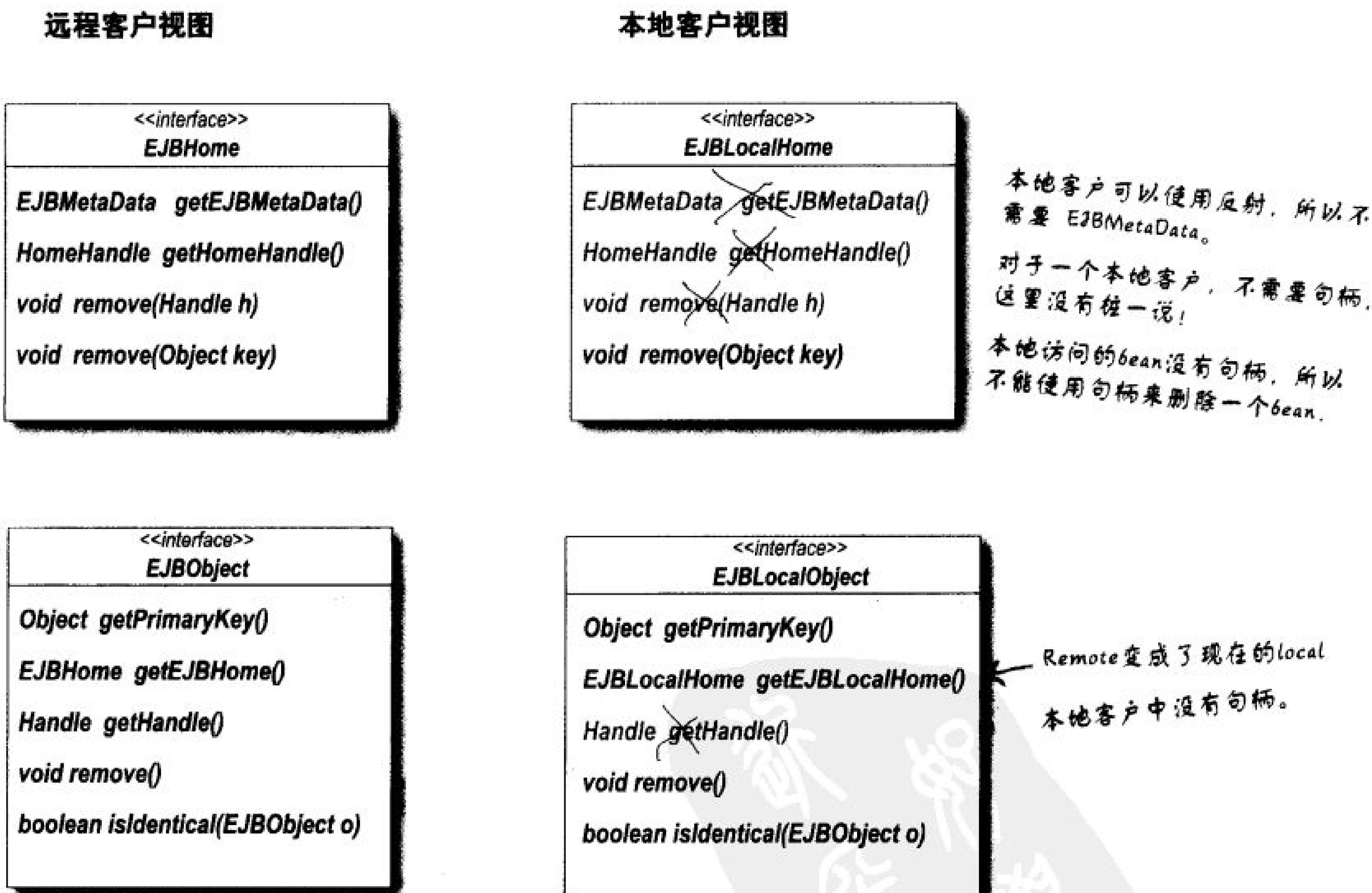
**也就是从这个home生成的bean。**

**这说明，在调用一个home删除方法时，必须惟一地标识你要删除的那个bean。对于实体bean，要用主键或者一个句柄来标识bean，对于会话bean，则必须使用句柄。**

**不过句柄只适用于远程home客户……**

# 远程接口与本地接口的比较

EJBObject接口和EJBHome接口的方法比EJBLocalObject和EJBLocalHome的方法多，因为有一些方法在本地上下文中是没有意义的。



## Sharpen your pencil

根据你对本地客户接口和远程客户接口之间差别的理解，确定下面的说法是否正确。对于有些说法，必须做一些推断和明智的猜测。

选出所有正确的说法：

- ❑ 要删除一个本地会话bean，惟一的办法就是通过组件接口。
- ❑ 实体bean可以通过一个本地home接口删除。
- ❑ 如果看到一个isIdentical()调用，这肯定是一个本地bean。
- ❑ 如果看到一个getHandle()调用，这肯定是一个远程bean。
- ❑ 如果客户在一个home方法上捕获了一个RemoteException异常，那么这个bean的home接口必定扩展了EJBLocalHome。
- ❑ 如果客户在一个业务方法上没有捕获RemoteException异常，那么这个bean的业务接口肯定扩展了EJBObject。
- ❑ 如果看到一个getEJBMetaData()调用，那么这个bean的组件接口必然扩展了EJBLocalObject。
- ❑ 如果在一个本地home上完成了一个JNDI查找，必须对JNDI返回的对象进行窄化。
- ❑ EJBLocalObject接口中有3个方法。
- ❑ EJBLocalHome接口中有2个方法。

# 编写本地客户接口

前面已经介绍了客户在一个本地接口里能够看到什么，下面来看作为一个bean提供者有哪些工作要做。换句话说，要为bean编写本地接口你必须做些什么。

**远程接口：**

```
package headfirst;
import javax.ejb.*;

public interface AdviceLocal extends EJBLocalObject {

   public String getAdvice();

}
```

扩展EJBLocalObject而非EJBObject

没有RemoteException!!

这个返回类型不必为RMI-IIOP兼容（不过，当然它也可能是RMI-IIOP兼容的，比如String）

**Home 接口：**

```
package headfirst;
import javax.ejb.*;

public interface AdviceHomeLocal extends EJBLocalHome {

   public AdviceLocal create() throws CreateException;

}
```

扩展EJBLocalHome而非EJBHome

这肯定是本地组件接口！

仍需要一个CreateException，但不需要RemoteException了。

**本地接口规则**

1. 导入 javax.ejb.*（或者使用完全限定名）。
2. 扩展EJBLocalObject（对于组件接口），或者扩展EJBLocalHome（对于home接口）。
3. 在组件接口中声明一个或多个业务方法。
4. 本地home接口中的所有创建方法都必须返回本地组件接口，并声明一个CreateException。
5. home或组件接口中声明的方法可以声明你自己的应用异常，这必须是编译器受查异常（也就是说，不能是RuntimeException的子类）。
6. 任何方法都不能声明RemoteException异常。

## bean可以同时有一个远程客户视图和一个本地客户视图，不过，一般不太可能这样。

前面提到过，本地接口对于客户视图来说是一种非常特殊的情况。本地接口是EJB规范2.0版本才引入的（本书所针对的就是这个版本），它的出发点是支持实体bean中的容器托管关系。不过，太多的顾客和开发商都要求bean还能有非远程接口，所以J2EE小组决定不仅实体bean可以有本地接口，会话bean也可以有本地接口。

但是不管你怎样选择，如果一个设计同时需要本地客户视图和远程客户视图，这一般是不大可能的。如果你的bean与另一个实体bean有一个容器托管关系（在关于实体的一章将详细介绍），那你就别无选择。这个bean必须本地展现。在这种情况下，要是说出于某种用途这个bean还要展现给远程客户，这是很难想象的。

只要知道同时提供两种客户视图是合法的就行了。

但是你千万不要把本地客户视图和远程客户视图混在一起乱搭配。

远程home接口只会给出远程组件接口（换句话说，就是远程EJBObject的桩）。本地home只会给出本地组件接口引用（即EJBLocalObject的一个常规的Java引用，堆内引用）。

不能把一个bean的本地接口和远程接口混在一起，即便给定的bean可以同时展现一个远程客户视图和一个本地客户视图，也不能把本地和远程接口乱搭配。

但是远程home只会给出远程组件接口引用（桩），而本地home只会给出本地组件接口引用（正常的Java引用）。

## Sharpen your pencil

使用AdviceLocal和AdviceHomeLocal的本地接口,把AdviceClient从一个远程客户改成本地客户。先不要翻到下一页看答案!!

```
import javax.naming.*;
import java.rmi.*;
import javax.rmi.*;
import headfirst.*;
import javax.ejb.*;

public class AdviceClient {

   public static void main(String[] args) {
      new AdviceClient().go();
   }

   public void go() {
      try {
         Context ic = new InitialContext();
         Object o = ic.lookup("Advisor");

         AdviceHome home = (AdviceHome) PortableRemoteObject.narrow(o, AdviceHome.class);

         Advice advisor = home.create();
         System.out.println(advisor.getAdvice());
      } catch (Exception ex) {
          ex.printStackTrace();
      }
   }
}
```

要做哪些修改?把修改的代码行写在下面:

# 客户接口中的异常：客户可能会得到哪些异常

远程接口中，每个方法都必须声明RemoteException异常。这说明，使用远程接口的客户必须为每个远程方法调用处理RemoteException。不过本地接口没有这个限制。本地客户接口中需要声明异常的方法只有create()和remove()方法（这是对会话bean而言的；实体bean还有一个查找方法会声明FinderException异常）。

我们会专门用一章来介绍EJB中的异常，所以这里先不做详细介绍，不过起码要了解：如果一个bean（或者容器）生成了一个运行时异常，远程客户会把这个异常看作是受查的RemoteException，而本地客户只会把它看成是一个非受查的EJBException。

除了接口方法声明的其他受查异常外（在EJB中，受查异常也称为应用异常），所有远程接口方法都可能抛出一个RemoteException，而本地客户接口方法总会抛出一个EJBException。所以远程客户必须把所有对home或组件接口方法的调用都包在一个try/catch里，而只有当接口方法声明了一个应用异常（这包括CreateException、RemoveException、FinderException和bean提供者在bean客户接口方法上所声明的其他异常）时，本地客户才会使用try/catch。

✓ 表示编译器受查异常（即非RuntimeException异常，也就是并非RuntimeException的子类）。

| | 远程客户视图 | 本地客户视图 |
|---|---|---|
| 所有方法 | ✓javax.ejb.RemoteException | javax.ejb.EJBException |
| 创建方法 | ✓javax.ejb.RemoteException<br>✓javax.ejb.CreateException | javax.ejb.EJBException<br>✓javax.ejb.CreateException |
| 删除方法 | ✓javax.ejb.RemoteException<br>✓javax.ejb.RemoveException | javax.ejb.EJBException<br>✓javax.ejb.RemoveException |

# 本地客户代码

将你在前一页的练习中得出的代码修改与下面的代码做个比较。为了有助于显示对home和组件接口的调用不是远程调用，这里的异常处理粒度更细。请注意，这里没有捕获RemoteException。

```
import javax.naming.*;
import headfirst.*;
import javax.ejb.*;
```

这里去掉了导入javax.rmi和java.rmi的语句

```
public class AdviceLocalClient {

    public static void main(String[] args) {
        new AdviceLocalClient().go();
    }

    public void go() {
        Object o = null;

        try {

          Context ic = new InitialContext();
          o = ic.lookup("AdvisorLocal");
```

你还是必须通过JNDI，按正常的方式完成查找，不过这一次你会得到堆上一个真正的Java对象（EJBLocalHome的一个实例）的引用，而不是远程EJBHomeobject的一个桩……

```
        } catch (NamingException nex) {
             nex.printStackTrace();
        }

        AdviceHomeLocal home = (AdviceHomeLocal) o;
        AdviceLocal advisor = null;
```

这是一个重大修改！没有窄化了！只用一个普通的类型强制转换（还是需要强制转换，这是因为JNDI查找的返回类型是Object，但是不必对它窄化，因为它不是一个桩）。

```
        try {

          advisor = home.create();

        } catch (CreateException cex) {
             cex.printStackTrace();
        }
```

create()方法还要声明一个CreateException异常，但是不再声明RemoteException。

```
        System.out.println(advisor.getAdvice());

    }
}
```

这个业务方法调用不再是一个远程方法调用。这只是一个正常的本地方法调用，而且由于getAdvice()没有声明任何异常，所以业务方法调用不用包装在一个try/catch中。

BRAIN POWER

## bean类里应该做哪些修改？

前面已经了解了把一个远程客户视图改成一个本地客户视图时接口要怎么修改，以及怎么修改客户代码。但是bean类本身怎么修改呢？你是怎么考虑的？如果要用一个本地客户视图而非远程客户视图来部署，bean代码需要修改吗？如果你打算同时用一个本地客户视图和一个远程客户视图来部署，又会怎么样呢？

对现在来说，先假设只关心bean的业务方法。在原来的bean类中，这个业务方法如下所示：

```
public String getAdvice() {
    System.out.println( in get advice );
    int random = (int) (Math.random() * adviceStrings.length);
    return adviceStrings[ random] ;
}
```

发现这个方法里有什么东西看上去是远程客户视图所特有的吗？对于本地客户是不是需要来点不一样的？

不，不要这么想。

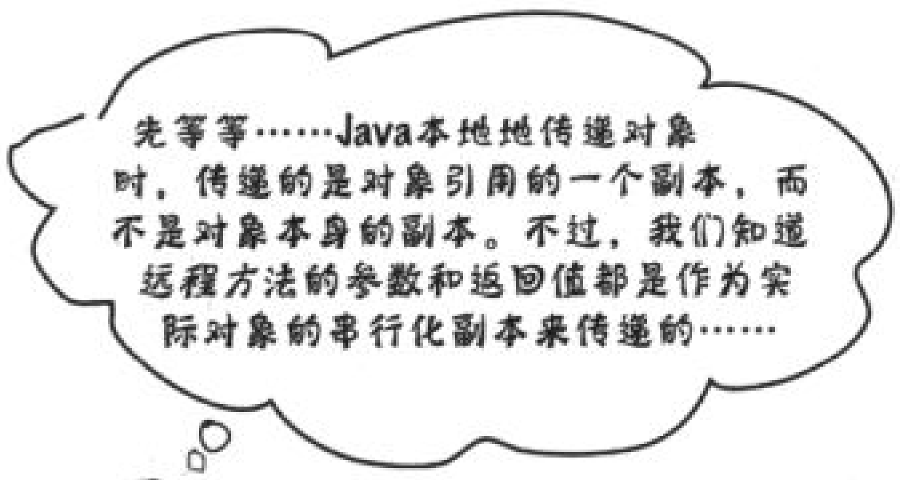
适用于远程方法的返回类型或参数肯定也能用于一个本地方法，所以不用做任何修改也可以（不过，返回类型是String好像有点太简单了）。不错，这里也有一个例外，要记住，根据bean法则，不能从一个本地接口方法返回bean的远程接口。

所以看上去（至少对于这个bean）不需要我们了解什么，甚至无需我们关心。就照这个bean的原样应该也能部署，而且bean并不知道它的客户是远程客户还是本地客户。

听上去很不错，对不对？既简洁，而且还面向对象。

不过再多想想看。假设有一个有更多复杂逻辑的bean。它有更多的业务方法，这些业务方法还有很多参数，甚至还可能需要对这些参数有所作用（有所修改）。

嗯……你能想到吗，如果这个bean知道客户是本地客户而不是远程客户，是不是会有些不同的处理呢？






在正常的本地方法调用中，Java会按值传递对象引用，即作为引用变量的一个副本来传递。

对象本身是不传递的。

不过，对于远程调用，对象本身会复制。

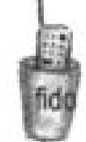
对于远程调用，所调用的方法总是作用于调用者的对象的一个*副本*。

对于本地调用，所调用的方法总是作用于调用者原来的对象，而*不是*一个*副本*！

# 远程和本地方法的参数

## 本地方法调用

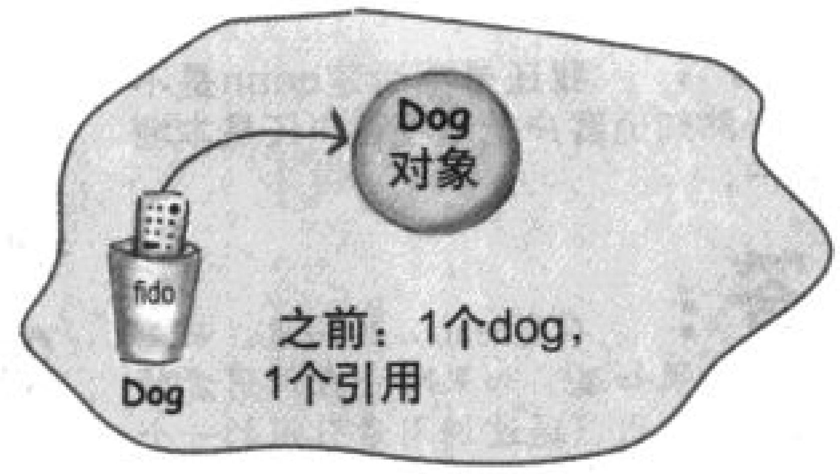
```
Dog fido = new Dog();
this.trainPet(fido);
```

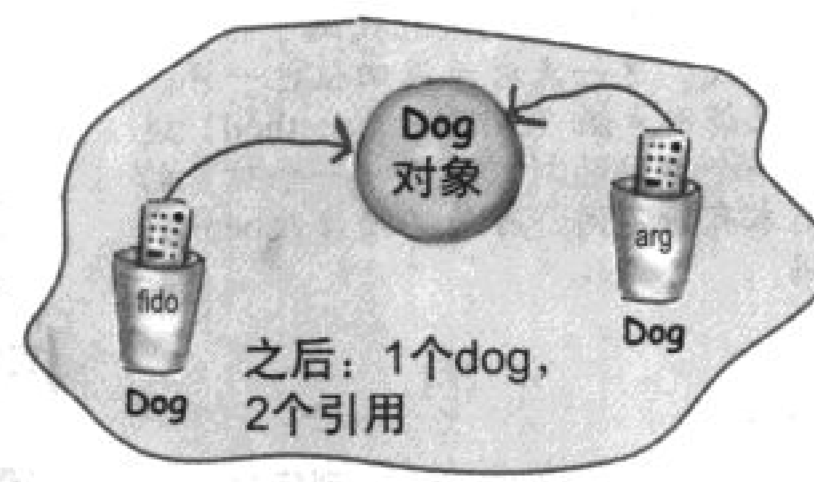



引用（遥控器）的一个副本，而非Dog对象的副本

```
void trainPet(Dog arg) { ... }
```

现在"arg"参数和"fido"变量是一样的副本。它们都引用了同一个Dog对象。




## 远程方法调用

```
Dog fido = new Dog();
remoteStub.trainPet(fido);
```

不是引用值！




这是实际Dog对象的一个串行化副本

```
void trainPet(Dog arg) { ... }
```




服务器上的Dog是客户上Dog的一个副本



## there are no Dumb Questions

**问：我还是不清楚bean是不是必然知道客户是远程客户还是本地客户。**

答：不是bean必然知道，而是可能必须知道。如果bean在调用者的对象上作用（通过调用者引用的一个副本）与在调用者对象的一个副本上作用有区别，你的bean代码可能就必须修改。这一点对于返回值也一样。如果调用方法（calling method）返回一个引用的副本和返回一个对象的副本有所不同，可能必须修改bean代码。

**问：不过，我觉得到底是基于本地客户视图还是远程客户视图来部署一个bean，这只是部署时要做的一个选择。**

答：不！不！不！你大错特错了。假设你确实编写了两组接口，一个用于本地客户视图，另一个用于远程客户视图。不错，也可以在部署时决定你想展现哪一个视图（或者二者都展现），但这只在bean代码不关心客户在哪里的情况下才可行。如果bean没有参数或返回值，这样的bean方法可能就是安全的，也就是不论客户以何种方式访问它（本地还是远程），这个bean方法都无需修改。

这里有一种解决方案，可以这样编写bean代码，假设bean总是得到一个副本，如果客户是本地的，则客户在传递对象之前先建立对象的一个副本（克隆）。或者，对于返回值，可以在bean把对象返回之前先建立它的一个副本。这样一来，bean就不必担心本地客户可能会修改bean的对象。

**问：难道只与参数和返回值有关吗？为什么不能先部署bean，再在部署时决定究竟是远程视图还是本地视图，这没有其他原因了吗？**

答：还有一个原因。这就是客户代码！即使bean不需要知道它的客户怎么访问它，但客户必须很清楚！如果一个客户编写为本地访问一个bean，而这个bean只有远程客户接口，这就不行，反过来也一样，如果把客户编写为远程访问，但只有本地客户接口，同样是不行的。

**问：为什么不行？**

答：客户必须提前知道它访问的是bean的远程客户视图还是本地客户视图，因为接口本身就不同。不管怎样，都不能交替地使用远程和本地视图，因为接口本身不一样。

客户要什么也不知道是不行的，因为bean的行为不同。要记住，远程客户必须处理RemoteException异常，还要把从查找得到的远程桩进行窄化！远程客户会看到bean客户接口中的方法，这些方法在本地接口里并没有。所以，比如说，远程客户可能想在本地组件接口上调用一个getHandle()方法，这个方法调用就绝对走不通。

本地客户中不会有代码来处理RemoteException异常或窄化桩。

这里的重点是，到底用远程客户视图还是本地客户视图来部署bean，这是一件大事。也是一件非做不可的事情，客户必须提前知道。

**问：能不能在本地接口上声明RemoteException异常绕过这个问题呢？能不能让组件接口（比如Advice）同时扩展Remote和EJBLocalObject……这样是不是就能建立一个既是远程又是本地的接口？如果客户总是处理RemoteException异常，而且总是做narrow()，会有什么不好呢？这样的话，客户就不用提前知道到底是本地客户视图还是远程客户视图了。**

答：还是不行。首先，根据bean法则，不允许在本地方法上声明RemoteException！所以你的服务器是不是允许你部署一个带本地接口又声明了RemoteException的bean，甚至这一点都很难说。而且也不能保证narrow()方法不会带来问题。除此以外，还有句柄，还有很多其他问题……所以还是别这么想了。




# 当容器



这一页上的每个代码段分别表示一个接口或一个客户的代码。你的任务是当一个容器，确定根据Java法则和bean法则，这些代码段是否合法。换句话说，即使代码能顺利编译，对容器而言可能仍是错误的，因为它可能没有遵循EJB规范的规则。这里假设你没有看到的所有一切都是合法而且正确的（也就是这些代码的上下文一切正常）。如果有问题，请确定问题是一个编译器错误，还是一个容器问题，并得出如何修正。

**A. 在一个本地客户中：**

```
public void go()  {
  Object o = null;
  try {

     Context ic = new InitialContext();
     o = ic.lookup( AdvisorLocal");

  } catch (NamingException nex) {
        nex.printStackTrace();
  }
  AdviceHomeLocal home = (AdviceHomeLocal) o;
  AdviceLocal advisor = null;
 // 更多内容
}
```

**B. 在一个远程客户中：**

```
public void go() {
   try {
    // 查找Advice bean，把它分配给advisor
   } catch (Exception ex) {
     ex.printStackTrace();
   }
   System.out.println(advisor.getAdvice());
}
```

**C. 在bean的home接口中：**

```
package headfirst;
import javax.ejb.*;

public interface AdviceHome extends EJBHome {

   public Advice create() throws CreateException;

}
```

**D. 在bean的组件接口中：**

```
package headfirst;
import javax.ejb.*;

public interface AdviceLocal extends EJBLocalObject {

   public String getAdvice();

}
```

## 要点

- 可以使用一个本地客户视图把bean展现给本地客户。
- 本地组件接口必须扩展javax.ejb.EJBLocalObject。本地home接口必须扩展javax.ejb.EJBLocalHome。
- 本地客户接口中的方法不声明RemoteException。
- 展现给远程客户的一些接口方法不会展现给本地客户。
- 本地客户无法得到句柄，因为句柄用于重新建立与远程对象的连接。
- 本地客户不必使用EJBMetaData，因为本地客户可以使用反射来了解EJB对象和Home对象的信息。
- 本地home接口只有一个remove()方法，这个方法取一个主键作为参数。在本地home接口中没有取句柄作为参数的remove()方法，因为本地客户视图不使用句柄。
- 由于本地home接口中惟一的remove()需要一个主键参数，所以本地会话bean客户不能使用bean的home来删除一个bean；本地会话bean客户只能在bean的组件接口上调用remove()。
- EJBLocalHome只有一个方法：这就是取一个主键作为参数的remove()方法，这是因为EJBHome中的getHomeHandle()、getEJBMetaData()和remove(Handle)方法不适用于本地客户视图。
- EJBObject中有但EJBLocalObject中没有的方法只有一个，这就是getHandle()。
- 使用本地客户视图时，参数和返回值是按值传递的。换句话说，它们以正常的Java方式来传递（对象按引用副本来传递，基本类型按值副本来传递）。
- 本地客户不需要窄化home引用，因为这是一个正常的Java引用，而不是一个远程对象的桩。
- 本地客户不需要捕获RemoteException异常，因为本地接口方法没有声明RemoteException。



**当心！**

**必须搞清楚是本地客户视图还是远程客户视图！**

要准备查看bean代码、客户代码或接口代码，明确你看的是一个远程客户视图还是本地客户视图。

二者的差别可能很微妙！

例如，你可能看到客户做了一个业务方法调用，而这个方法的参数是一个不可串行化的对象。因为你认为方法是合法而且正确的，这样就能知道，这肯定是一个本地接口，因为如果是一个远程方法，把不可串行化的对象作为参数或返回值传递是不行的。

还有另外一些技巧可以区别出bean是本地的还是远程的，这包括：

(1) 客户没有窄化Home。

(2) 客户没有处理业务方法调用上的任何受查异常。

(3) 客户在bean的home上调用remove(Handle)。



## 练习答案



# 当容器

### A. 在一个本地客户中：

```
public void go()  {
  Object o = null;
  try {

     Context ic = new InitialContext();
     o = ic.lookup("AdvisorLocal");

  } catch (NamingException nex) {
        nex.printStackTrace();
  }
  AdviceHomeLocal home = (AdviceHomeLocal) o;
  AdviceLocal advisor = null;
 // 更多内容
}
```

A是对的。因为它是本地方法，不需要窄化Home引用，只需要一个类型强制转换就够了。

### B. 在一个远程客户中：

```
public void go() {
  try {
   // 查找Advice bean, 把它分配给
   // advisor
  } catch (Exception ex) {
    ex.printStackTrace();
  }
  System.out.println(advisor.getAdvice());
}
```

编译器错误！getAdvice()是一个远程方法，但它没有处理RemoteException。如果把最后一行移到try块里面，就可以了。

### C. 在bean的home接口中：

容器错误！EJBHome是一个远程接口，规则要求必须在接口中的每个方法上声明RemoteException异常。编译器并不在意，但是容器很在乎。如果没有声明这个异常，等到部署过程中的某个时刻，就会出现失败。

```
package headfirst;
import javax.ejb.*;

public interface AdviceHome extends EJBHome {

   public Advice create() throws CreateException;

}
```

需要RemoteException（请看导入语句，没有导入java.rmi.*）。

### D. 在bean的组件接口里

D是可以的……这是一个本地组件接口，它不需要在业务方法上声明任何异常。

```
package headfirst;
import javax.ejb.*;

public interface AdviceLocal extends EJBLocalObject {

   public String getAdvice();    不错，不需要声明异常

}
```

# COFFEE CRAM
# 模拟测验

**1** 以下哪些功能是会话bean的远程和本地home接口都提供的？（选出所有正确的答案。）

❏ A. 创建一个会话对象

❏ B. 删除一个会话对象

❏ C. 得到一个会话对象的EJBMetaData

❏ D. 得到一个会话对象的句柄

---

**2** 查找一个会话bean的远程home接口时，要创建一个合法的home接口引用，必须有哪几步？（选出所有正确的答案。）

❏ A. 会话上下文必须得到窄化，再对窄化的结果做类型强制转换

❏ B. JNDI查找的结果必须得到窄化，再对窄化的结果进行类型强制转换

❏ C. 初始上下文必须得到窄化，再对窄化的结果进行类型强制转换

❏ D. JNDI查找的结果必须类型强制转换为一个初始上下文，然后进行窄化

---

**3** 给定一个远程客户“R”，它有会话bean“A”和“B”的合法引用，再给定A是B的一个本地客户，下面哪些说法是正确的？（选出所有正确的答案。）

❏ A. R 不能把它的A引用传给B

❏ B. A 不能把它的B引用传给R

❏ C. A 不能调用B上的方法

❏ D. B 不能调用R上的方法

**4** 比较两个会话对象时，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 使用isIdentical()方法，来自同一个home的无状态会话bean总返回true
- ❑ B. 使用isIdentical()方法，来自同一个home的有状态会话bean总返回true
- ❑ C. isIdentical()方法只能用于远程对象引用
- ❑ D. 使用equals()方法，来自同一个home的无状态会话bean肯定返回true
- ❑ E. 使用equals()方法，来自同一个home的有状态会话bean肯定返回true

**5** 关于会话bean，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. bean提供者必须在无状态和有状态会话类中编写方法public void remove()
- ❑ B. 本地客户可以通过调用bean home上的一个方法来删除会话bean
- ❑ C. 组件接口中的remove()方法只能由远程客户使用
- ❑ D. 要让EJBHome删除一个会话bean，客户必须提供bean的句柄

# COFFEE CRAM
## 模拟测验答案

**1** 以下哪些功能是会话bean的远程和本地home接口都提供的？（选出所有正确的答案。）

☑ A. 创建一个会话对象

☐ B. 删除一个会话对象 — remove()方法

☐ C. 得到一个会话对象的EJBMetaData — 本地home里没有

☐ D. 得到一个会话对象的句柄 — 本地home里没有

会话bean不能通过一个本地home接口来删除，因为只有一个取主键作为参数的

（规范：57）

**2** 查找一个会话bean的远程home接口时，要创建一个合法的home接口引用，必须有哪几步？（选出所有正确的答案。）

☐ A. 会话上下文必须得到窄化，再对窄化的结果做类型强制转换

☑ B. JNDI查找的结果必须得到窄化，再对窄化的结果进行类型强制转换

☐ C. 初始上下文必须得到窄化，再对窄化的结果进行类型强制转换

☐ D. JNDI查找的结果必须类型强制转换为一个初始上下文，然后进行窄化。

**3** 给定一个远程客户“R”，它有会话bean“A”和“B”的合法引用，再给定A是B的一个本地客户，下面哪些说法是正确的？（选出所有正确的答案。）

☐ A. R 不能把它的A引用传给B

☑ B. A 不能把它的B引用传给R

☐ C. A 不能调用B上的方法

☑ D. B 不能调用R上的方法

不能把一个本地引用交给一个远程客户，A通过一个本地引用看到B

**4** 比较两个会话对象时，下面哪些说法是正确的？（选出所有正确的答案。） （规范：65–66）

- [x] A. 使用isIdentical()方法，来自同一个home的无状态会话bean总返回true
- [ ] B. 使用isIdentical()方法，来自同一个home的有状态会话bean总返回true
- [ ] C. isIdentical()方法只能用于远程对象引用
- [ ] D. 使用equals()方法，来自同一个home的无状态会话bean肯定返回true
- [ ] E. 使用equals()方法，来自同一个home的有状态会话bean肯定返回true  equals()的行为没有指定

**5** 关于会话bean，下面哪些说法是正确的？（选出所有正确的答案。）

- [ ] A. bean提供者必须在无状态和有状态会话类中编写方法public void remove()  — 是ejbRemove()，而不是remove()
- [ ] B. 本地客户可以通过调用bean home上的一个方法来删除会话bean  — 本地home只有一个取主键作为参数的remove()
- [ ] C. 组件接口中的remove()方法只能由远程客户使用  — 不，只能由本地客户使用
- [x] D. 要让EJBHome删除一个会话bean，客户必须提供bean的句柄

— 但对于EJBLocalHome并非如此

# 4 会话bean生命周期

## 当个会话bean




**会话bean要创建和删除。** 如果你是一个无状态bean，那你很幸运。因为有状态bean的一生都操纵在一个无情的客户手上。有状态bean是在一个客户的坚持下创建的，它的存在只是要为这个客户服务。作为一个有状态bean，要想不被删掉，最多只能寄希望于客户崩溃或者客户忘记调用remove()；这样，容器可能需要花一段时间才能发现你已经没用了。不过，啊哈，相比之下无状态bean的生活真是帅呆了！泳池、阳伞、饮料，一切都这么美好，这里没有乏味、没有单调，因为你要面对那么多形形色色的客户。当然，就算是无状态bean，也会因为遭遇一个非受查异常而死掉……

# OBJECTIVES

## 会话bean

| | 标准大纲： | 具体含义： |
|---|---|---|
| 3.1 | 有关会话bean的说法或示例，区别哪些是正确的，哪些是不正确的，这包括会话状态、SessionBean接口和创建方法。 | 只要是与会话bean有关，都会有这个要求，所以你一定要全面了解、熟练掌握，必须知道无状态bean和有状态bean生命周期的所有细节、SessionBean的容器回调方法、bean类中必须编写哪些代码，还要知道怎么从一个bean的EJBContext得到bean。掌握了这些要求，就能轻松地面对其他有关会话bean的要求了。 |
| 3.2 | 明确会话bean中的ejbPassivate方法如何使用，它有怎样的行为，这包括容器和bean提供者各自担负的职责。 | 首先，你必须知道钝化只用于有状态会话bean。无状态会话不通过钝化就可以回到bean池中。所以，ejbPassivate()绝对不会在一个无状态bean上调用。其次，你得知道，在ejbPassivate()中，你（bean提供者）的职责是确保当这个方法结束时你的bean要准备好钝化，而且钝化可能涉及串行化，也可能不涉及。再者，你要知道ejbPassivate()结束时你的实例变量所处的状态。而且，尽管这个要求里没有明白地提到ejbActivate()，但你也要了解ejbActivate()是怎么工作的。 |
| 3.3 | 对于以下各个行为，明确相应的接口和方法：获取会话bean的远程home接口，获取会话bean的本地组件接口，确定会话bean的调用者是否有一个特定的角色，允许实例把当前事务标记为回滚，获取UserTransaction接口，准备好实例以便钝化后重用，删除之前先释放资源，明确bean实例组件接口的调用者，新事务开始时得到通知，当前事务结束时得到通知。 | 你必须会分析对所需行为的描述，知道可以调用哪个方法，以及在哪个接口上调用才能得到这种行为，例如，如果要找出客户的安全主体，就应该在SessionContext接口上调用getCallerPrincipal()方法。<br><br>要回答这些问题，必须了解三个关键接口中的每一个方法：SessionBean（这是你的bean要实现的接口）、SessionContext（你的bean有这个接口的一个引用）和SessionSynchronization（这是有状态bean可以实现的一个可选接口）。在这一章中，除了SessionSynchronization外，我们会介绍有关会话bean接口的所有其他内容，SessionSynchronization将在关于事务的一章介绍。 |

第3章要求

## 会话bean

| 标准大纲： | | 具体含义： |
| --- | --- | --- |
| 3.4 | 将有关用途和功能的描述与相应的会话bean类型匹配，即无状态会话bean有哪些功能，有状态会话bean有哪些功能，哪些功能是无状态会话bean和有状态会话bean都具有的。 | 必须知道无状态会话bean和有状态会话bean之间的区别，这包括：在哪些情况下要选择无状态会话bean而不是有状态会话bean，或者在哪些情况下要选择有状态会话bean而不是无状态会话bean。你必须知道的bean规则包括：在一个无状态会话bean中只能有一个创建方法，而且这个创建方法必须是无参数的；仅有状态bean可以实现SessionSynchronization，如果容器崩溃，无论是无状态会话bean还是有状态会话bean都无法幸存，无状态bean不会钝化，而且无状态bean的创建和删除不受制于客户。 |
| 3.5 | 给定一组有关会话bean的职责，找出哪些是会话bean提供者的职责，哪些是EJB容器提供者的职责。 | 必须知道规范中哪些做了保证，而哪些不能保证。例如，容器不允许对一个会话bean进行并发访问（如果客户A正在使用一个bean，客户B调用这个bean上的一个方法时就会得到一个异常）。另外有些东西是不能保证的，比如容器没有要求必须把一个EAR文件中的一个bean作为另一个EAR中一个bean的本地客户。你要很清楚哪些类和方法要由你来实现（bean、ejbRemove()和ejbCreate()方法等等），而哪些是由容器实现的［句柄、home和组件对象、桩（对于远程客户接口），以及getEJBObject()方法等等］。这个要求中大多数内容都与3.1、3.2和3.6有重叠。 |
| 3.6 | 给定一组需求，找出其中会话bean类、远程组件接口、远程home接口、创建方法、业务方法、本地组件接口和本地home接口的需求分别是哪些。 | 这个要求有关于bean法则——也就是根据规范必须遵守的规则，不过编译器可能对此不做要求，也就是说，如果你没有遵循bean法则，编译器也不会在乎。例如，对home中的每个创建方法，都必须有一个与之匹配的ejbCreate()方法，如果没有这样一个匹配的ejbCreate()方法，编译器也不会报错。另外，必须知道本地接口方法不能声明一个RemoteException异常，但是远程接口方法必须声明这个异常。此外，还必须知道远程接口方法的对象类型参数是作为对象串行化副本传递的，而本地接口方法的对象参数则作为引用副本发送。 |

# 探究会话bean的生命周期

请把自己想像成一个会话bean。对你来说什么是重要的？在你的一生中有哪些关键的时刻？你对客户需要有哪些了解？对容器又需要有哪些了解？你要从容器得到些什么？

下面再假想你是一个客户。你的目的是什么？对于容器如何工作你需要知道些什么？再假如你是一个容器，为了管理一个bean的生与死，你要做些什么？什么时候做？

如果你是bean提供者，要想编写出能工作的有效代码，你要对bean的生命周期知道些什么？如果你还希望代码很高效，又要知道些什么呢？

在这一章中，我们会探访会话bean的整个世界，而且会从不同的视角来分析。这一章的组织采用了一种面向对象方法，多个层次层层深入，层层细化。所以，如果开始几页让你觉得层次过高，或者不会回答我们所留的问题，你也不要着急。等这一章介绍完了，你肯定会了解具体的细节。

开始介绍之前，先给大家介绍一下我们的演员：

**客户**



**目标：在bean上调用业务方法**



**bean**

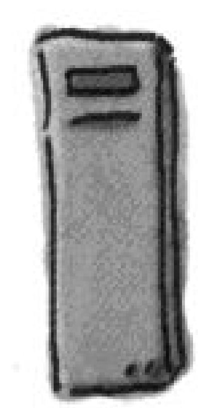
**目标：理解它人生的真正意义，来自哪里，有什么作用，以及怎么结束……**

**bean提供者**



**目标：编写可部署、表现好的bean。他很好地掌握了bean API，知道应该把哪些代码放到哪里。**

**容器**



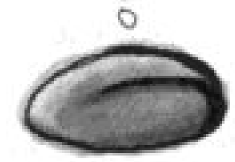
**目标：这是bean的上帝。它管理着bean的生与死，会截获对bean的所有方法调用。**

我在home和组件接口桩上调用方法来创建、使用和删除bean。总有一天，我要亲手杀了bean提供者（开个玩笑），他总是不给我正确的桩。



客户

我是一个对象，要由我运行业务方法，我同时还是一个bean。作为bean，我要注意在我一生中最关键的一些时刻，比如构造（很麻烦）、bean创建（非常重要）、钝化（打算睡觉）、激活（醒来）和删除（死掉，我不希望死，如果你是我，你也不想死吧）。每次出现这些特殊时刻时，容器就会告诉我。这些事件大多数只发生一次（构造、创建、删除），但激活、钝化还有业务方法可以反复发生。



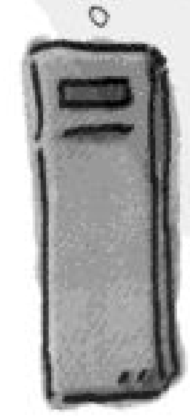
bean

我来写bean代码。除了业务方法外，我还必须实现容器回调（就是出现bean一生中关键时刻的时候容器要调用的方法）。另外我还必须编写与home中create()方法相匹配的ejbCreate()方法。我的工资太低了，他们甚至连把好椅子都不给我……



bean提供者

我来构造bean，是我让bean真正成为一个bean，具备了bean性（有了特殊的bean功能），我会在bean上调用创建方法，如果它是一个有状态bean则还会传入客户参数。我来决定bean是不是要睡觉和醒来，什么时候睡，什么时候醒，而且只有我能杀死（删除）bean。我来决定是否在一个bean上调用业务方法，怎么调用，并且加入一些服务，比如安全和事务。对bean来说，我就是上帝。



容器

# 你还记得这些是怎么做的吧？

## 得到和使用一个有状态会话

这里有一个前提，就是客户已经在JNDI完成了查找，而且得到了home桩，所以下图显示的是在此之后的情况。换句话说，它显示了客户怎么得到一个bean，然后再怎么使用它。不过，这个图的层次有点过高，所以在这一章后面还会更详细地介绍。例如，我们会说明第2步（home对象得到客户的create()调用）和第4步（home建立bean）之间到底发生了什么。考试要求你相当清楚“建立bean”涉及哪些工作。

**创建bean**

1 客户在home桩上调用create()。

2 Home对象得到create()调用。

3 Home/容器为这个bean建立EJBObject（组件接口）。

4 Home/容器建立bean。

5 Home向客户返回EJBObject（组件接口）桩。

**使用bean**

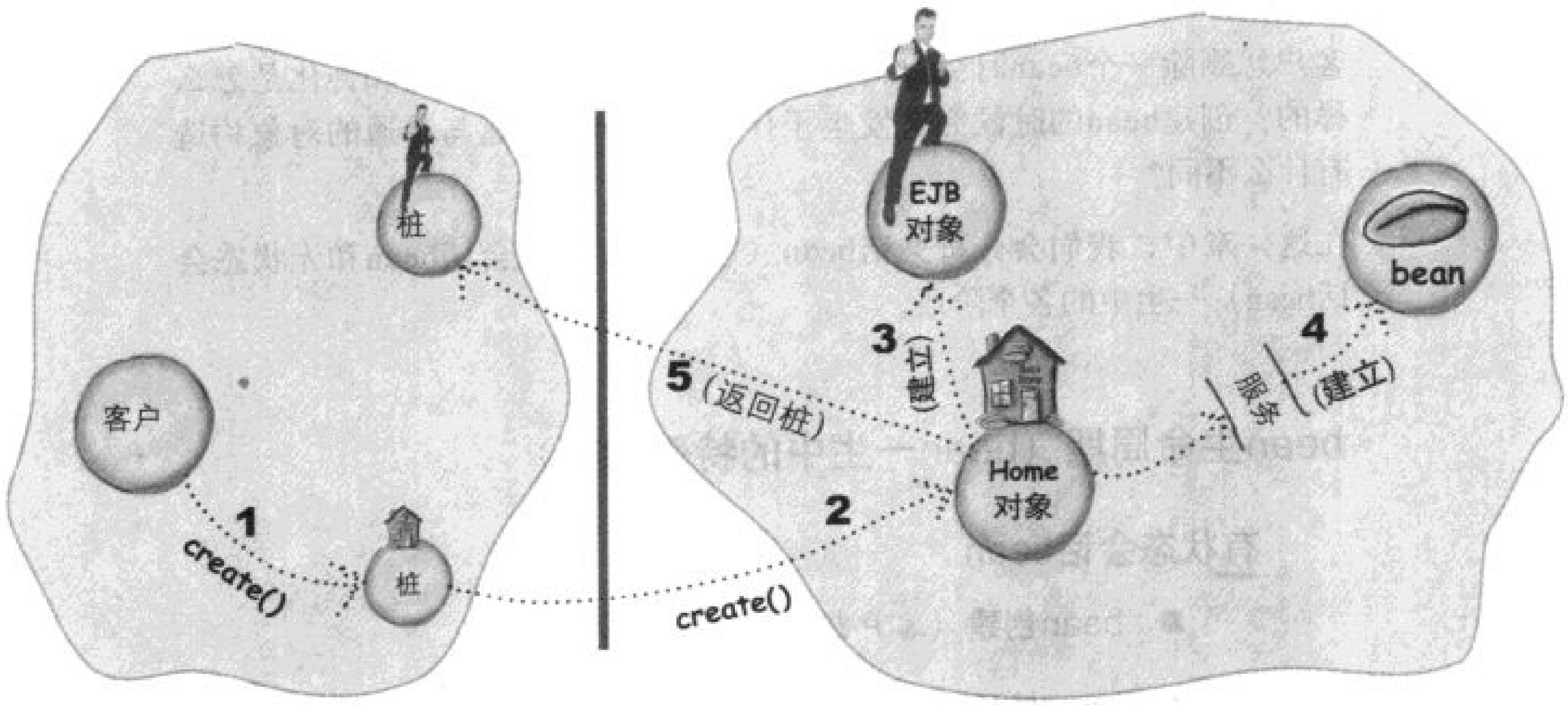
6 客户在组件接口桩上调用一个业务方法(getAdvice())。

7 EJBObject得到getAdvice()调用。

8 容器介入，在bean上调用getAdvice()。

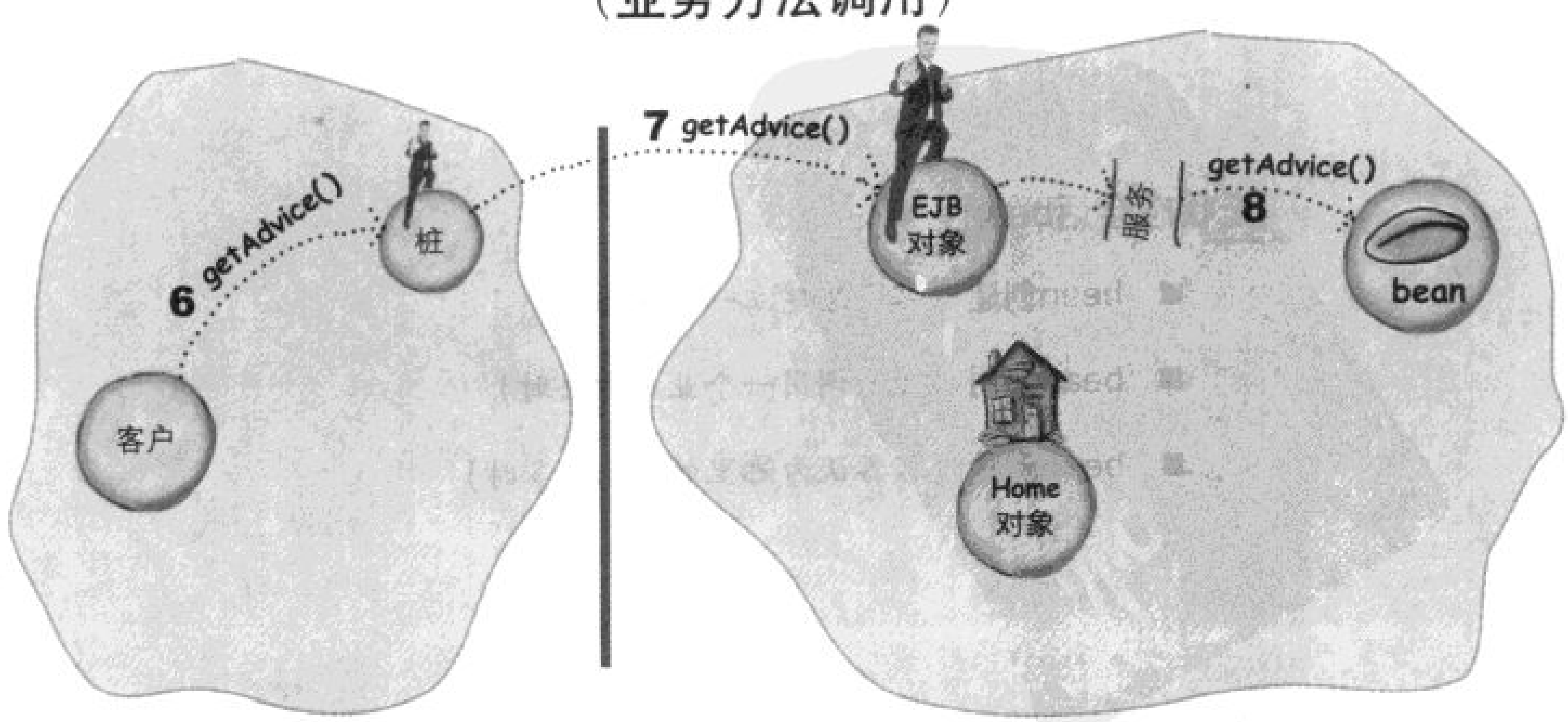
# 有状态会话bean

## 创建




## 使用

## (业务方法调用)



# 除了创建和业务方法外，很显然，bean的生命周期远不止这些……

客户想删除一个bean时会如何呢？有状态会话bean的激活和钝化是怎么样的？创建bean的时候究竟发生了什么？bean的创建与普通的对象构造有什么不同？

在这一章中，我们会分析会话bean（这包括有状态会话bean和无状态会话bean）一生中的各个阶段。

## bean生命周期（bean一生中的特殊时刻）

### 有状态会话bean

- **bean创建**（客户想要一个bean时）
- **bean使用**（客户调用一个业务方法时）
- **bean钝化**（让bean睡觉，来节省资源）
- **bean激活**（bean醒来，为客户的一个业务方法调用提供服务）
- **bean删除**（客户用完这个bean时，或者bean超时）

### 无状态会话bean

- **bean创建**（容器想建立一个bean时）
- **bean使用**（客户调用一个业务方法时）
- **bean删除**（容器认为池里的bean太多时）

## 容器回调
## bean一生中的特殊时刻……




### 作为一个bean提供者，你必须在bean类中实现容器回调！

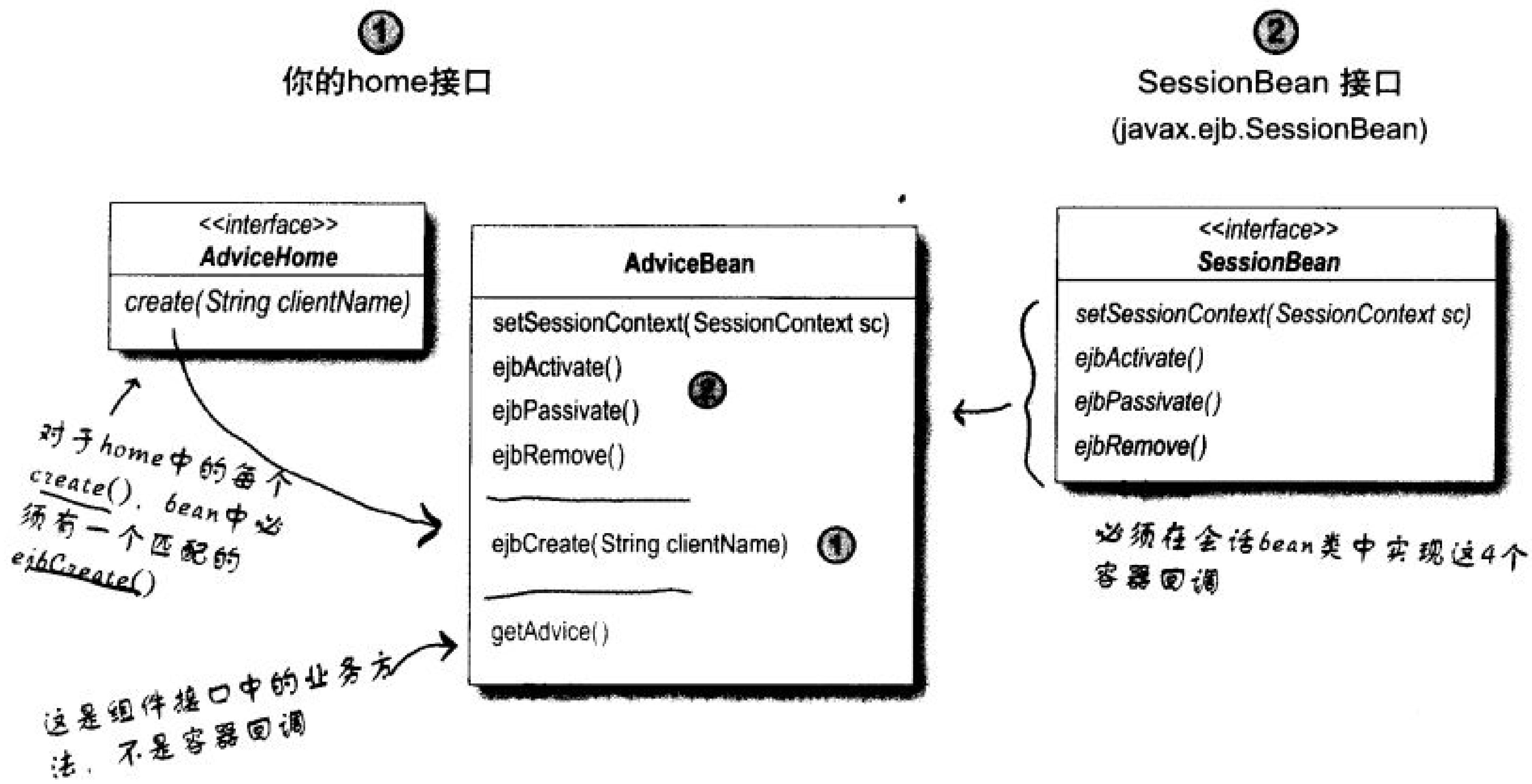
出现一个特殊时刻的时候，bean是不知道的，直到容器调用了bean的某个容器回调方法时它才会知道。这些回调方法是容器了解的一些特殊方法，你（作为bean提供者）必须在bean类中实现这些方法。可以把容器回调看作是一种事件处理程序。

容器回调来自两个地方：一个是bean的home接口，另一个就是会话bean类必须实现的SessionBean接口。

对于一个会话bean，与home相关的容器回调是一些创建方法，这些创建方法（ejbCreate()）与bean的home接口中声明的各个create()方法相匹配。

SessionBean接口声明了4个容器回调，这些回调方法的作用分别是：为bean提供bean上下文、激活bean、钝化bean和删除bean。下一页将具体分析这些方法。

# 容器回调来自两个地方：




**对于任何会话bean，肯定至少有5个容器回调**，其中4个来自SessionBean接口实现，另外至少有一个ejbCreate()对应于home接口中声明的create()方法。

对于无状态会话bean，就只有5个容器回调，因为无状态会话bean只有一个create()方法。而且，由于无状态bean的create()总是一个无参数方法，所以可以知道，无状态bean的ejbCreate()肯定是无参数的。

对于有状态bean，要记住，至少要有一个create()，但也可能有多个，这包括重载的无参数create()方法和有参数的create<something>方法。例如，有状态bean home可能有3个创建方法：create()、create(Strings)和createBigCustomer(Strings)。这就说明，这个bean会有3个来自home的容器回调：ejbCreate()、ejbCreate(String s)和ejbCreateBigCustomer(String s)。

# 实现容器回调

这个AdviceBean代码是完全合法的。它能够编译和运行。现在，在回调方法中还不需要加任何东西，因为bean的业务逻辑并不依赖于bean一生中其他时间发生的任何事情。可以注意到，我们保存了从Set Session Context()容器回调得到的SessionContext参数，但是这个代码里并没有用它。在这一章的后面，你就会知道为什么想把这个上下文赋给一个实例变量，还会了解到在什么情况下要在这些回调方法里放上东西。

```
package headfirst;
import javax.ejb.*;

public class AdviceBean implements SessionBean {

    private SessionContext context;
    private String name;


    public void ejbActivate() {

    }

    public void ejbPassivate() {

    }

    public void ejbRemove() {

    }

    public void setSessionContext(SessionContext ctx) {
       context = ctx;
    }


    public void ejbCreate(String clientName) {
        name = clientName;
    }

    public String getAdvice() {
       return "Advice for" + name + ": there is no spoon.";
    }
}
```

必须实现SessionBean

实例变量，保存从容器回调方法得到的上下文

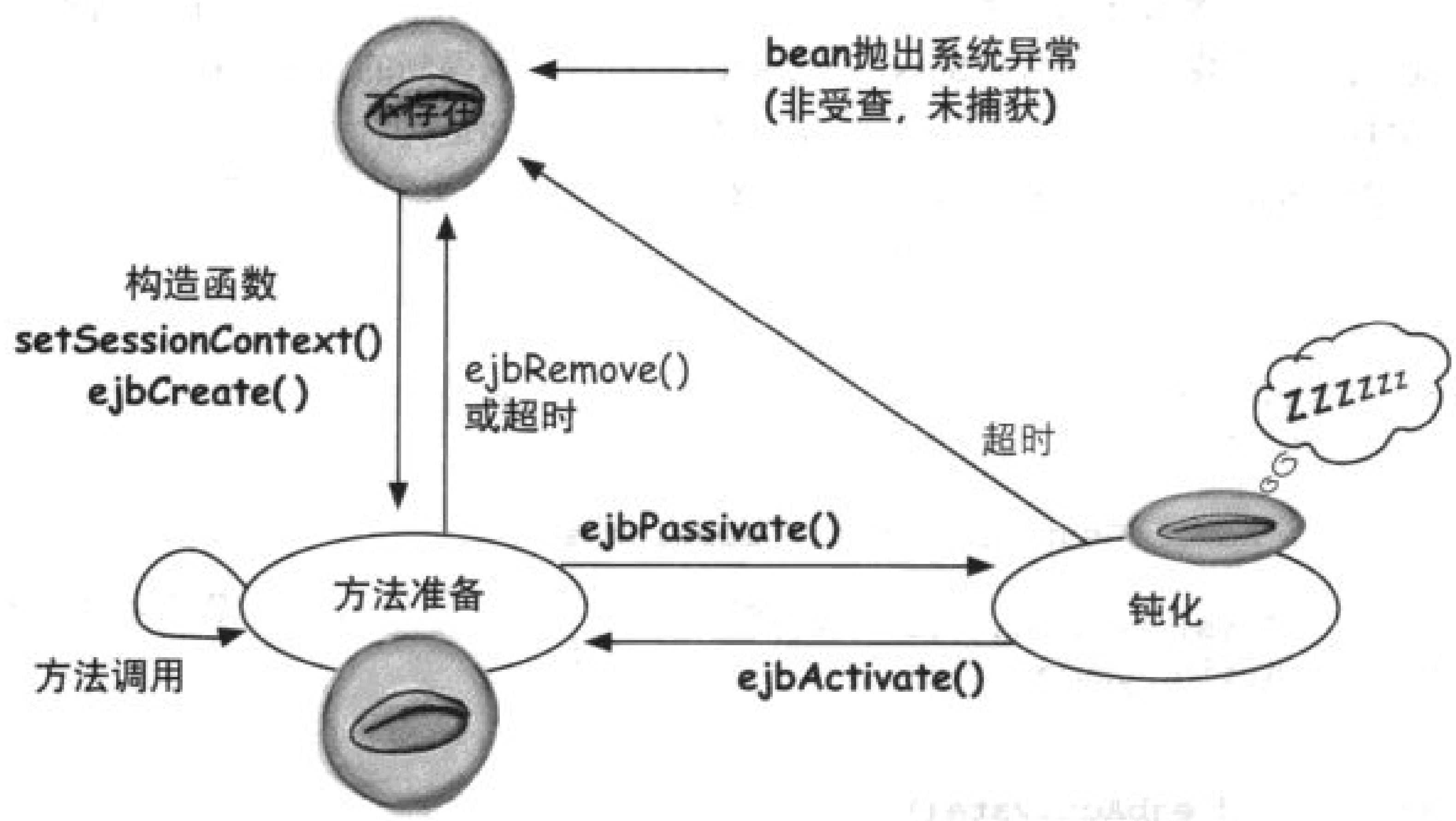
你**必须**实现SessionBean的这4个方法

这个方法必须与home接口的创建方法匹配

这是组件接口的业务方法……这不是一个容器回调

## 什么时候涉及容器回调?

### 对于有状态会话bean，bean什么时候从一个状态变迁到另一个状态。




**① 不存在** bean成为一个bean（或者甚至成为一个对象）之前，它是不存在的。规范对于这一点做了很充分的说明，这样就不至于对bean存在之前有些什么产生混淆。所以“不存在”状态实际上是即使bean还不存在也能进入的状态。会话bean从这个状态移出时，会运行bean的构造函数，然后运行setSessionContext()，最后会运行ejbCreate()方法。如果发生以下某个情况，bean还会返回到这个状态，这包括：bean超时，bean抛出一个系统异常，或者客户调用了remove()。如果客户调用了remove()，或者bean还处于方法准备（活动）状态时出现超时，bean就会得到一个ejbRemove()调用。

**② 方法准备** 处于方法准备状态的bean要么在执行一个客户的方法，要么在等待客户做另外一个业务方法调用。有状态bean处于这个状态时，可能在一个活动事务中，也可能不在一个活动事务中，这取决于部署事务设置和调用者的事务状态。不过，如果bean确实处在一个事务中，那么这个bean不能钝化。

**③ 钝化** 钝化的bean会临时保存在某种辅存中，从而在客户调用的间歇节省资源。钝化可能是串行化，不过容器可以采用任何途径来完成钝化，只要表现得像串行化就行（对此只有一个很小的例外，我们将在这一章后面介绍）。尽管钝化的bean不再是堆中一个活动的对象，但是当客户调用一个业务方法时，容器还能重新激活这个bean（调用ejbActivate()）。如果一个钝化的bean超时，这个bean可以直接死掉，不用先重新激活。

## 必须详细看看状态变迁

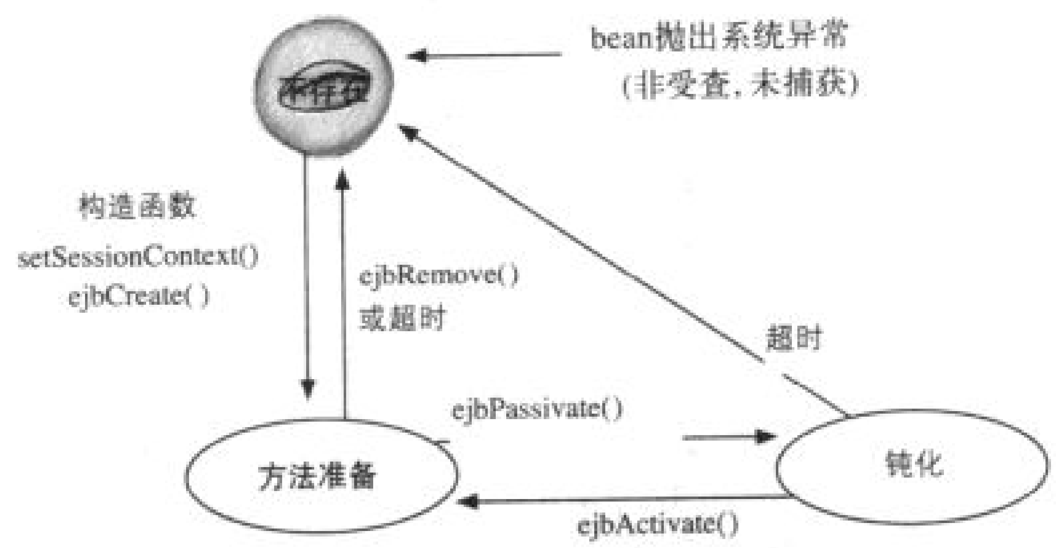
前面已经了解了整个状态图，下面需要细化了，看看一个有状态会话bean从一个状态变迁到另一个状态时究竟会发生什么，以及为什么会这样。

**从不存在 到 方法准备**

构造函数

setSessionContext()

ejbCreate()



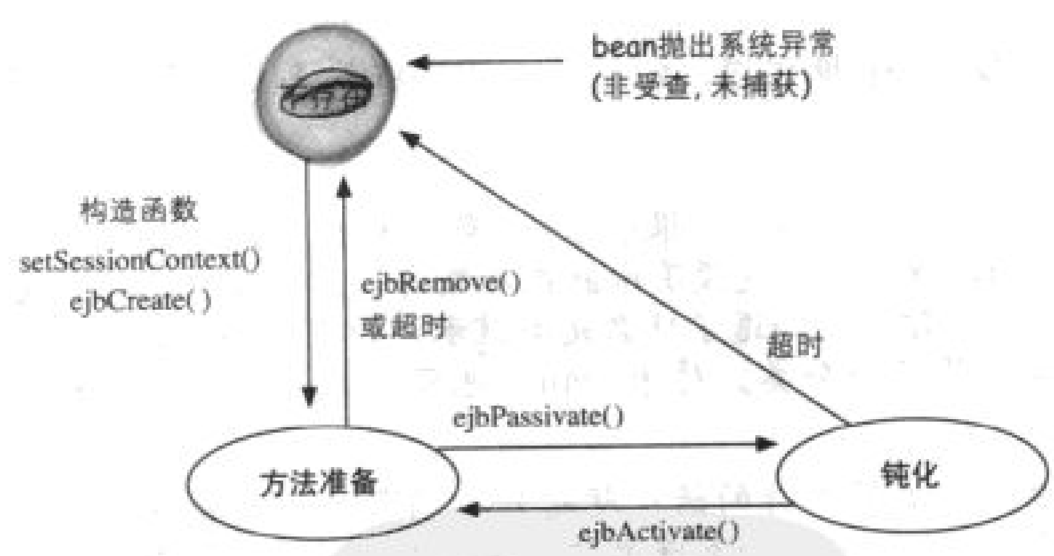

---

**从 方法准备 到 钝化 以及**

**从 钝化 到 方法准备**

ejbPassivate()

ejbActivate()




---

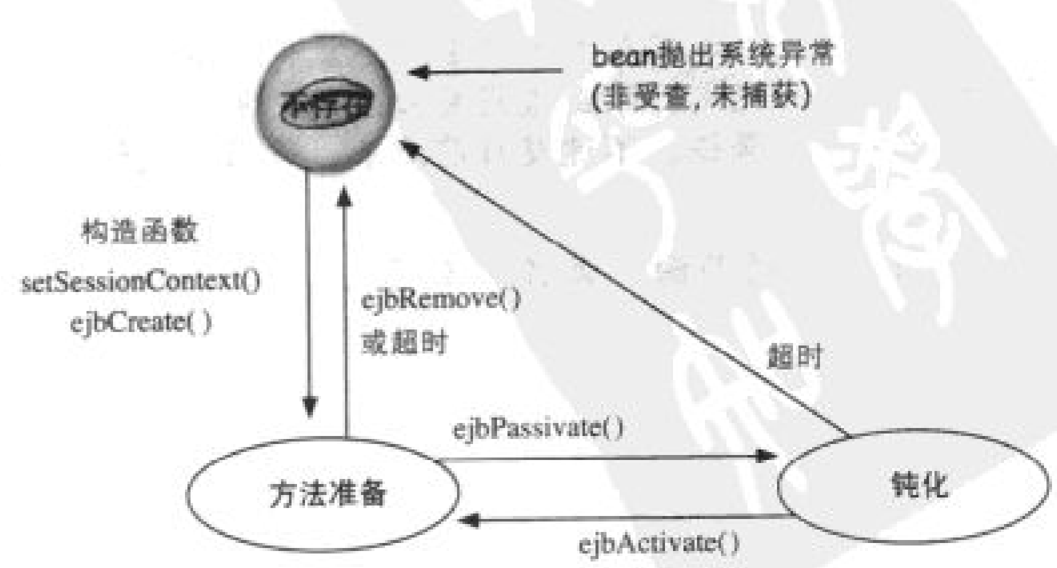
**从 钝化 或 方法准备**

**到 不存在**

超时，系统异常

或

ejbRemove()



## there are no Dumb Questions

**问：** **如果经常让这些方法体为空，那么为什么不像事件处理程序那样建立一些适配器类，让这个适配器类来实现接口的所有方法呢？bean类不能扩展一个实现了SessionBean接口的类，这有什么原因吗？**

**答：** API里没有为SessionBean实现建立适配器类（也就是说，没有一个实现了SessionBean接口所有方法的类）。不过并没有说不能自己建立这样一个适配器类，你完全可以这样做。但是要记住，对于实际的bean，有可能在其中某个方法或者某些方法里有具体的代码（而不是空）。而且，你甚至可能会使用一个支持bean的IDE，它没准会以某种方式为你建立（填写）这些方法。

不过，如果确实建立这样一个通用的bean，它有SessionBean的所有方法，而且一般都可以从这个通用bean扩展，那会很方便。特别是对于无状态bean，因为就算无状态bean从来不会调用ejbActivate()和ejbPassivate()，也必须实现这两个方法！（无状态会话bean不可能钝化；有关这个内容在本章后面还会详细介绍。）

**问：** **我记得曾听说过，企业bean不支持继承！这是怎么回事？**

**答：** 哈哈……这是一个很常见的误解。不错，好像是这样。EJB支持常规的Java类继承，但是没有bean继承概念。你可能会问："这到底有什么区别？"你已经知道了什么是类继承，所谓类继承，在Java中这就是一个类扩展另一个类。对于bean，也可以像其他类一样进行类继承。

但是bean继承（如果支持的话）就是另一码事了，这表示一个bean类可以扩展另一个bean类，而且继承的不仅是类中可继承的成员，还要继承"父bean"的bean性（beanness）。哪一种bean性可以继承呢？（假设将来确实决定支持这种bean继承，这也不是不可能的。常规的非企业bean就确实支持bean继承。）

一种想法是，让bean子类继承其超类的部署描述文件设置，然后用一个小得多的不完整的部署描述文件来覆盖想要修改的设置。这样看起来很棒；是不是真的很棒不能保证，不过就目前看来，这只是我们的一个小小的梦想。

不管怎么样，只要对你的面向对象设计有意义，就完全可以让你的bean扩展另一个类。

## Sharpen your pencil

要参加这个考试，必须非常清楚SessionBean接口里有哪些方法，所以从现在开始，我们就要开始记了。看看你能不能记住与这里描述的行为相匹配的方法名。在这里的描述中，我们会给出很明显的提示，因为这是你第一次记这些方法，不过等到真正的考试时，题目可就没这么明显了……

1. 当客户告诉容器他已经用完了一个有状态会话bean时会调用这个方法。这让bean很不高兴：

________________________

2. 让bean睡觉以临时节省资源的时候调用这个方法：

________________________

3. 把原来睡觉的bean叫醒以为一个业务方法提供服务时，调用这个方法：

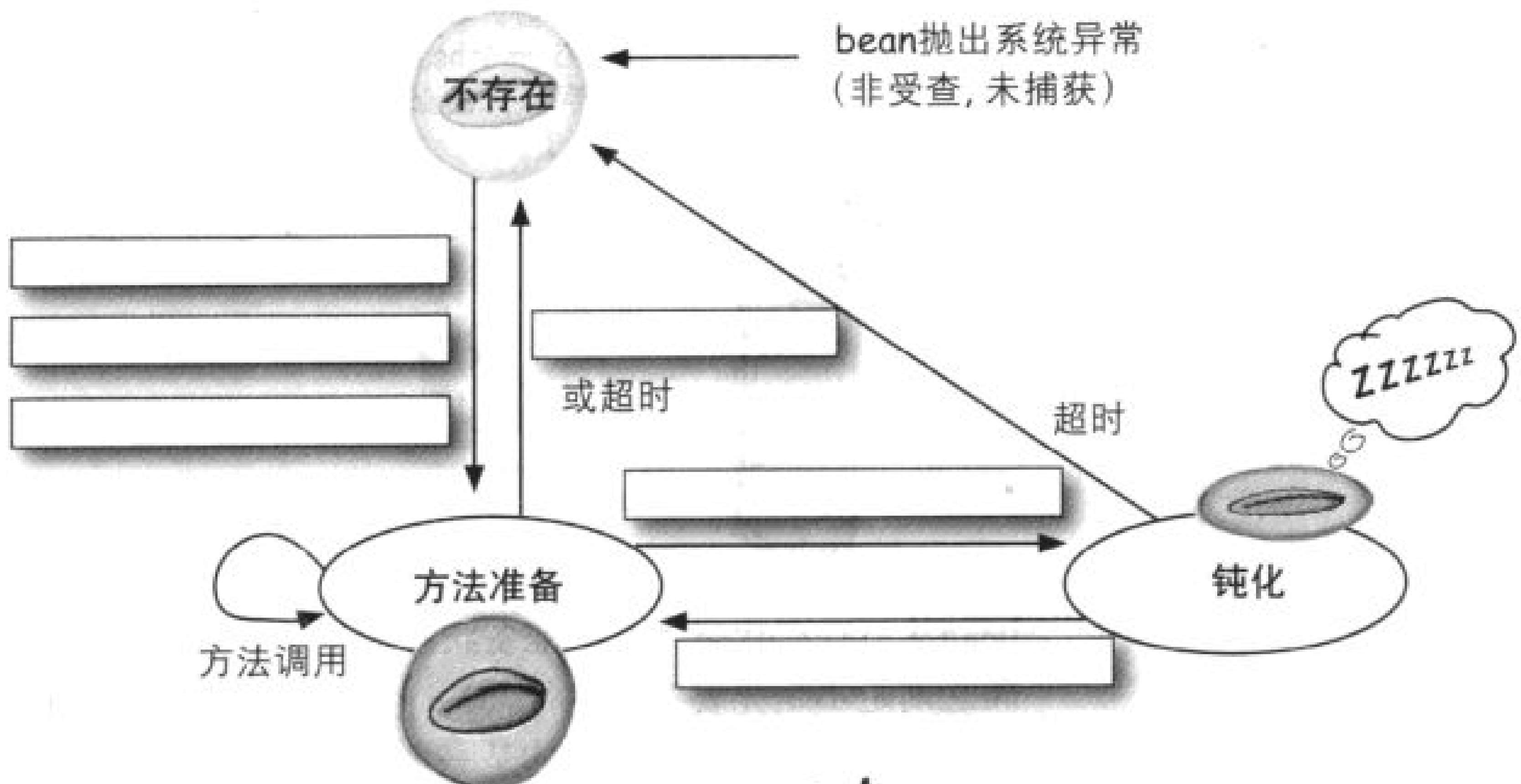
________________________

4. 在bean生命的最初调用这个方法，这个时候容器交给bean一个引用，也就是这个bean与容器的一个特殊连接的引用：

________________________

## Sharpen your pencil

填空，填上少了哪些方法。如果不能准确地写出方法名也没关系；只要指出这个有状态会话bean的状态图中各个变迁上发生了什么就可以。

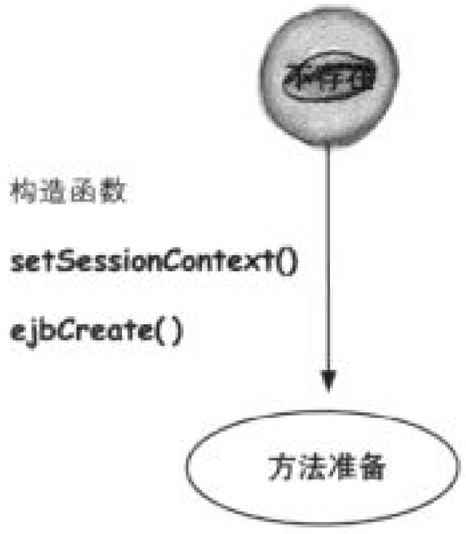



### 必须掌握这个生命周期

我们为什么要这么反反复复地讲这个生命周期，如果你真的参加了考试，就会知道这是为什么了，而且肯定会对我们感激不尽，因为你确实必须很清楚地掌握这个内容。

但是即使不参加考试，作为一个bean开发人员这也很重要，而且因为这个内容如此基础，你肯定想把它记住。这和那些有关API的问题不同，对于那些API问题，你会想："不是可以查文档吗？"但这些问题应该不查文档就很清楚。

即使你的IDE会为你在代码中放上容器回调，你也要知道这些容器回调的含义，还要知道在每个回调方法里应该做什么，不应该做什么。




setSessionContext()和ejbCreate()在会话bean的一生中只发生一次，所以不要浪费了！另外也别太快做事情……在构造函数中做任何bean工作都为时太早。

# bean创建：对象什么时候变成bean




bean从不存在状态到方法准备状态（是不是感觉像要爬……爬之前是不是先要存在才行？），先要从一个构造函数函数开始。但是构造函数建立的是一个对象，而不是bean。要成为一个bean，必须给这个对象一些bean性。

对象变成一个bean时，它会得到bean性带来的所有特有权限，比如能够得到有关客户的安全信息，或者可以在JNDI中这个bean的特殊空间里查找特殊的部署时性质，再或者是从容器管理的一个池中得到一个JDBC连接。

### 为什么要关心这些创建细节？

这是因为，在调用构造函数和create()方法之间的某个时刻，bean处在一种Schroedinger bean*状态。你可能有一些bean初始化代码，比如要得到一个JDBC连接，或者查找另一个bean的引用，这些初始化代码如果在bean人生中太早运行，就会出问题。要编写好的bean代码，这说明你必须知道在哪个时刻对象会成为一个正式的bean，而这对你（开发人员）又意味着什么。

*我们并不打算解释完整的Schroedinger（薛定谔）引用，这里只会指出它涉及到猫科动物和亚原子物理。要了解更多这方面的信息，没准我们以后会写一本《Head First Quantum Physics》，你可以拿来看看。这可不是开玩笑的，我们一直在说正经的。

# “bean性”意味着什么？

bean从一个对象　　变成一个正式的bean时，发生了什么？






**不要把bean上下文和JNDI上下文混为一谈！**

你开始可能觉得这很清楚，但同时又惊恐地发现，“上下文”（context）这个词有多重含义！
bean的上下文（如SessionContext、EntityContext或MessageDrivenContext）是bean用来从容器获得信息的特殊引用（它扩展了EJBContext）。

这与JNDI上下文没有任何关系。要记住，JNDI上下文是JNDI虚目录树上的一个特定“节点”。InitialContext就是JNDI上下文，它是你进入这个树结构的入口点。换句话说，你要从这里开始在树中导航。

只要看到“上下文”这个词，都请仔细看看问题的上下文，明确问题的上下文到底指的是bean的EJBContext还是JNDI上下文。小心了！

## ① SessionContext引用

bean的上下文（有时称为EJBContext，这是指SessionContext的超接口）是bean到容器的惟一生命线，利用bean上下文，bean能做很多事情，如得到有关调用客户（calling　client）的安全信息、要求一个事务回滚、得到bean自己的home或EJB对象的一个引用，等等。

## ② 特殊的JNDI上下文

每个bean都会得到它自己特有的JNDI上下文，在这里，bean可以找到许多东西，包括资源管理器连接工厂（这些对象可以为你提供诸如数据库等资源的连接）、其他bean和一些部署时常量值（你可以使用这些部署时常量值来定制bean的变量）。在第11章还会介绍更多这方面的内容，第11章会分析企业bean环境（规范称之为bean的特殊JNDI上下文）。

## ③ 访问bean和资源

如果得到了一个数据库的连接，这并不意味着你总能使用这个连接。作为一个bean，很重要的一方面就是能够访问一个资源（如数据库），或在另一个企业bean上调用方法。

# 特殊的bean工作



## ① SessionContext引用

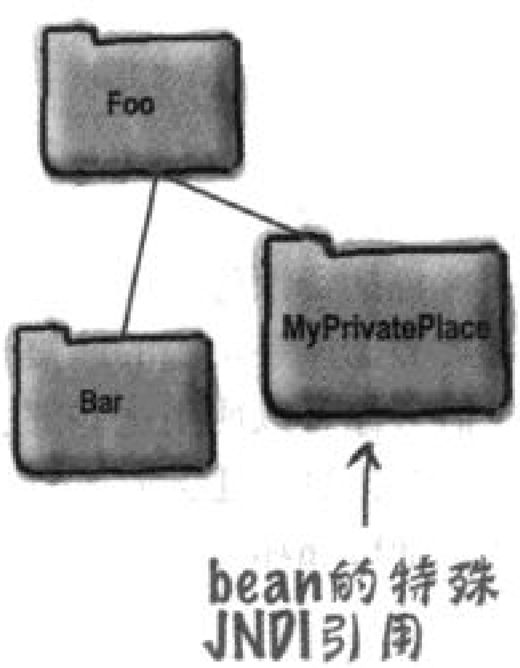
**用bean上下文能做的事情：**




- 得到home的一个引用
- 得到EJB对象的一个引用
- 得到有关客户的安全信息
- 要求一个事务回滚 (CMT)
- 确定事务是否已经置为回滚 (CMT)
- 得到一个事务引用，并在上面调用方法 (BMT)

## ② 特殊的JNDI上下文

**用特殊的JNDI上下文能查找的东西：**




- 另一个bean的引用
- 一个资源管理器连接工厂（如数据库）的引用，可以用这个引用得到一个资源连接，如一个数据库连接
- bean的部署时常量值（就是性质之类的东西），这些常量值由部署人员设置，可由bean在运行时作为变量查找和使用
- 一个“受管对象”资源（这通常表示一个JMS目标）的引用

## ③ 访问……

- 另一个bean
- 资源管理器（如数据库）



## 并不总能使用bean性的所有方面……

如果你成为了一个完整意义上的bean，这并不一定意味着你可以做上一页所列的所有bean工作（查找一个资源、让一个事务回滚，等等）。

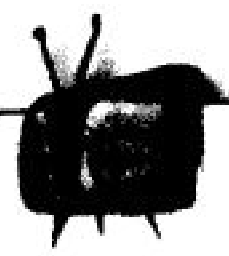
你到底能做什么？取决于你是哪一类bean（会话bean、实体bean还是消息驱动bean）、你的事务状态以及你正在调用哪一种方法，所能做的有所不同。

例如，如果你是一个有状态会话bean，那你就是由于客户在你的home上调用了create()而创建的。如果你是一个有状态bean，而且正在调用你的ejbCreate()方法，那么这只有一个原因：客户要求你创建。而这意味着，通过向EJBContext（如SessionContext）要客户安全信息，你就可以找出客户是谁。

但是如果你是一个无状态bean，那么你的创建与任何特定客户的请求并无关联（在后面几页你就会了解这一点）。实际上，如果容器愿意，即使没有任何客户，容器也可以先在池中创建大量无状态bean。而这意味着无状态bean无法在ejbCreate()中得到有关客户的安全信息。因为根本没有客户！这里只有容器，调用容器回调并不是客户调用的一部分，而且我们也不认为容器是一个客户。容器应该是老板，而不是客户。

另外有些bean工作只能在特定的情况下做，也就是客户认为bean处于“一种有意义的事务上下文”中时。例如，如果一个方法没有事务，就不能访问数据库。

有些bean工作是互斥的。如果你在使用容器托管事务（container-managed transaction，CMT）（见第9章），就无法让EJBContext向你提供你自己的事务对象。另一方面，如果你在使用bean托管事务（bean-managed transaction，BMT），那么可以请求一个事务对象，但是不能让你的上下文回滚事务。你必须自己来回滚事务，这里可以使用从你的上下文得到的事务对象。



**必须知道什么时候可以做特定的bean工作，而什么时候不可以。**

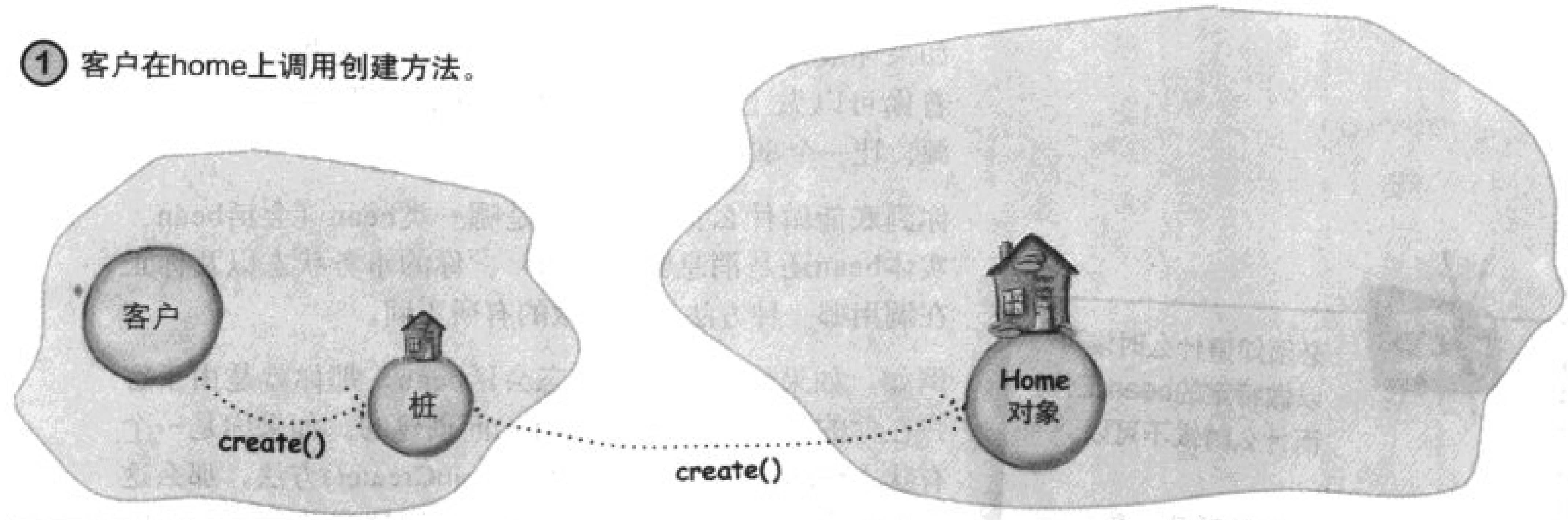
你可能会遇到这样的题目，给你一段代码或一个场景，让你看一个方法，并问你：在这种情况下，这个方法能不能做某个特定的bean工作（Bean Thing）。

例如，可能给你看一个bean的setSessionContext()方法，并问你这个方法能不能访问一个资源（如一个数据库）。（这是不行的！因为此时还没有处在一个“有意义的事务上下文”中。）

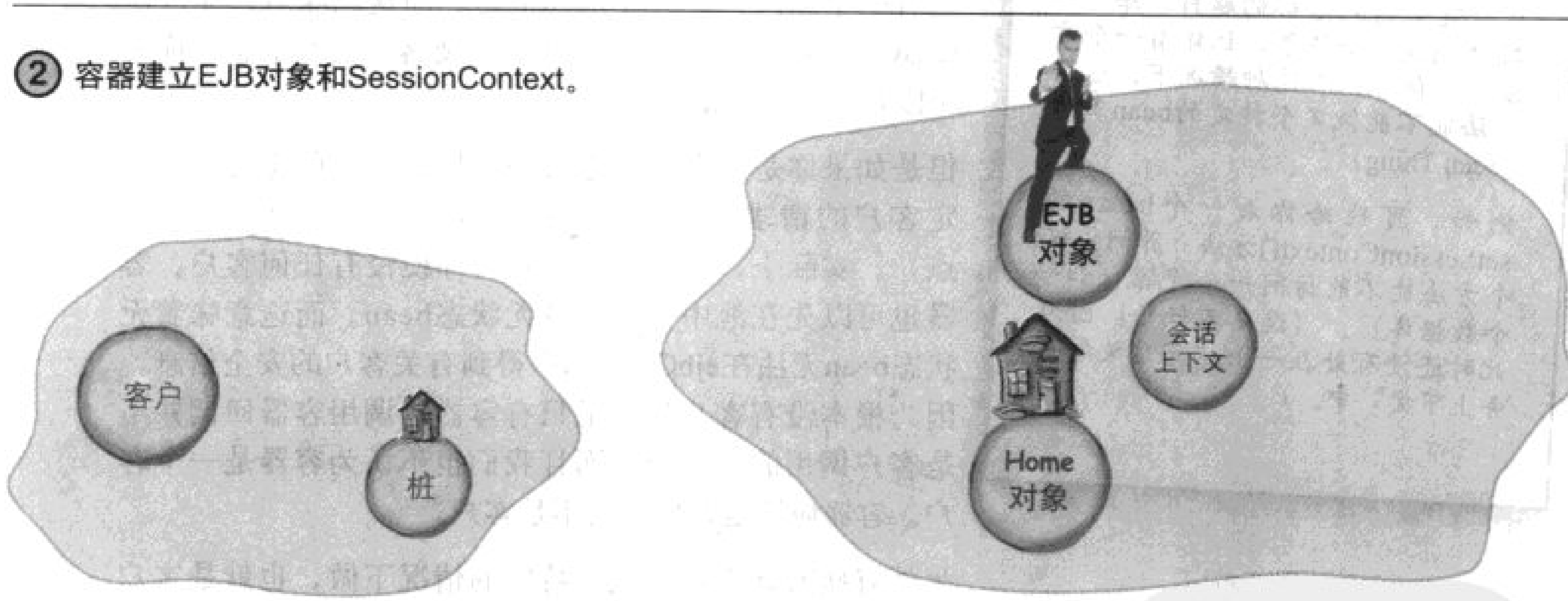
# bean创建概述

(有状态会话bean)

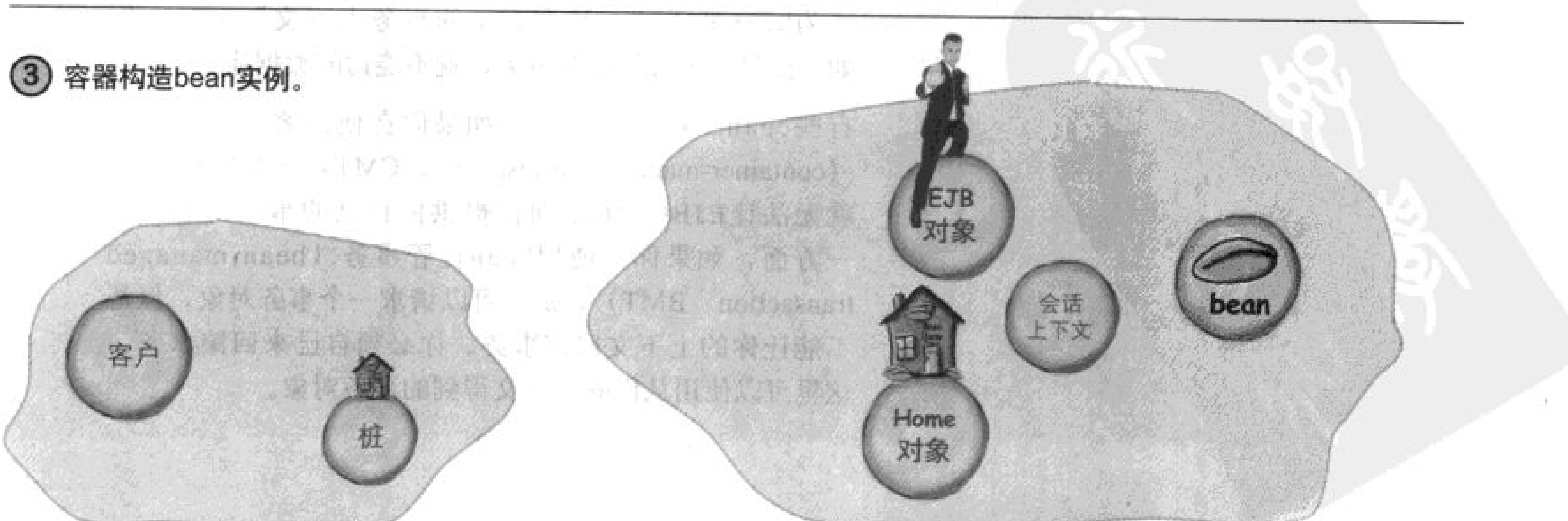
① 客户在home上调用创建方法。



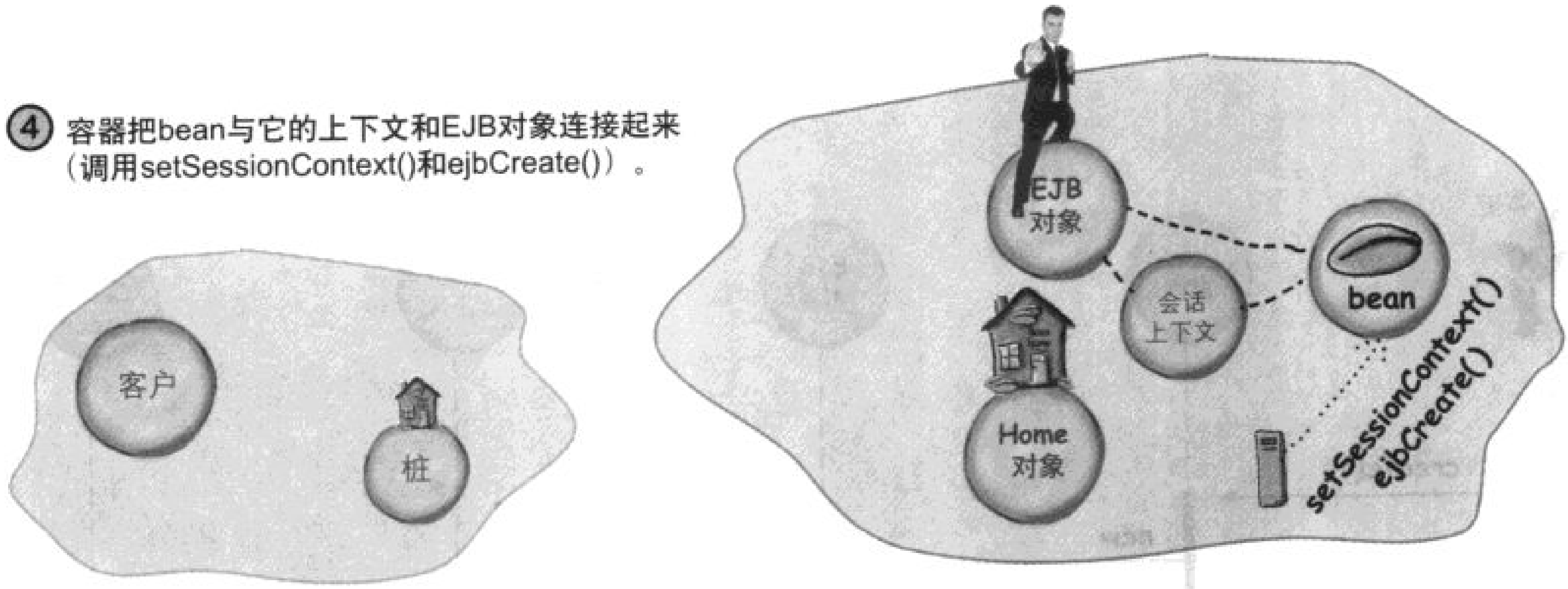

② 容器建立EJB对象和SessionContext。




③ 容器构造bean实例。



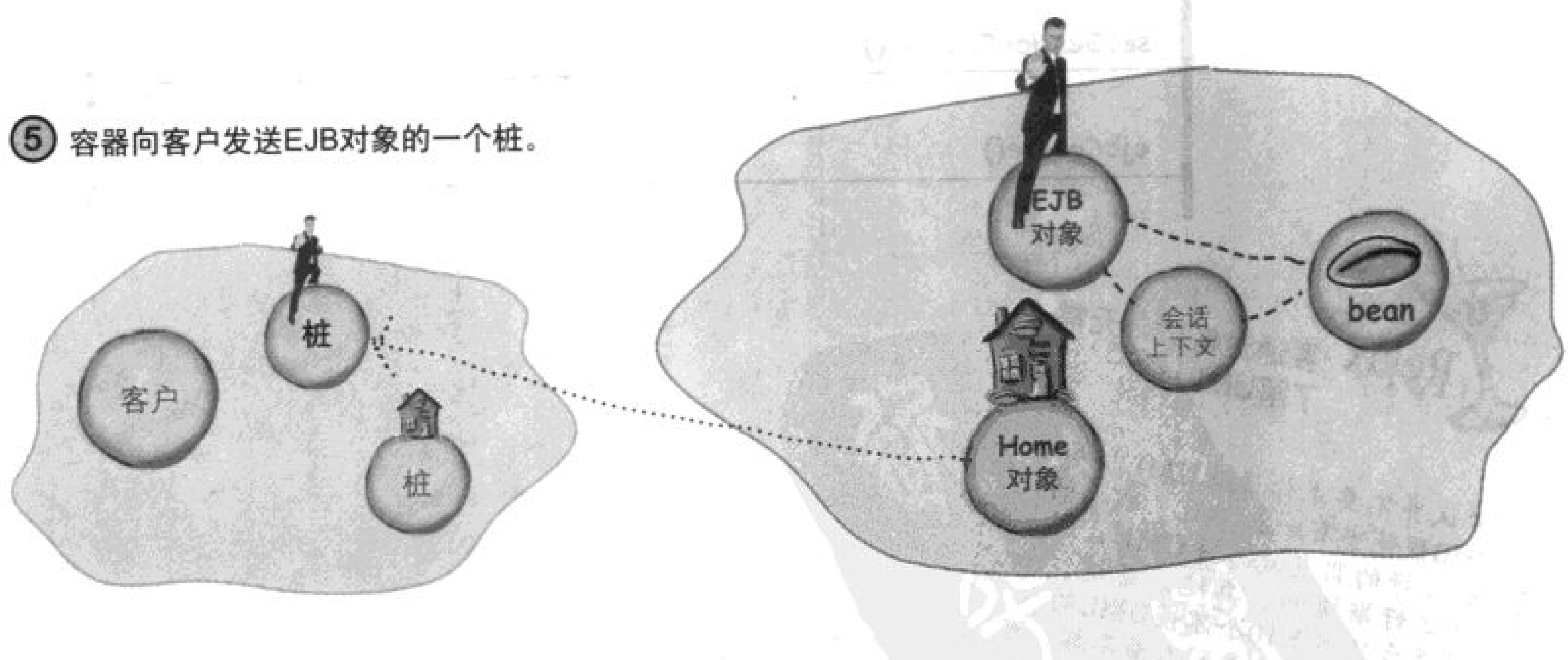
④ 容器把bean与它的上下文和EJB对象连接起来（调用setSessionContext()和ejbCreate()）。




这个连接发生在容器在bean上调用setSessionContext()和ejbCreate()的时候。一旦ejbCreate()返回，这个bean就准备好“投入工作”（换句话说，可以为客户发出的业务方法调用提供服务了）。

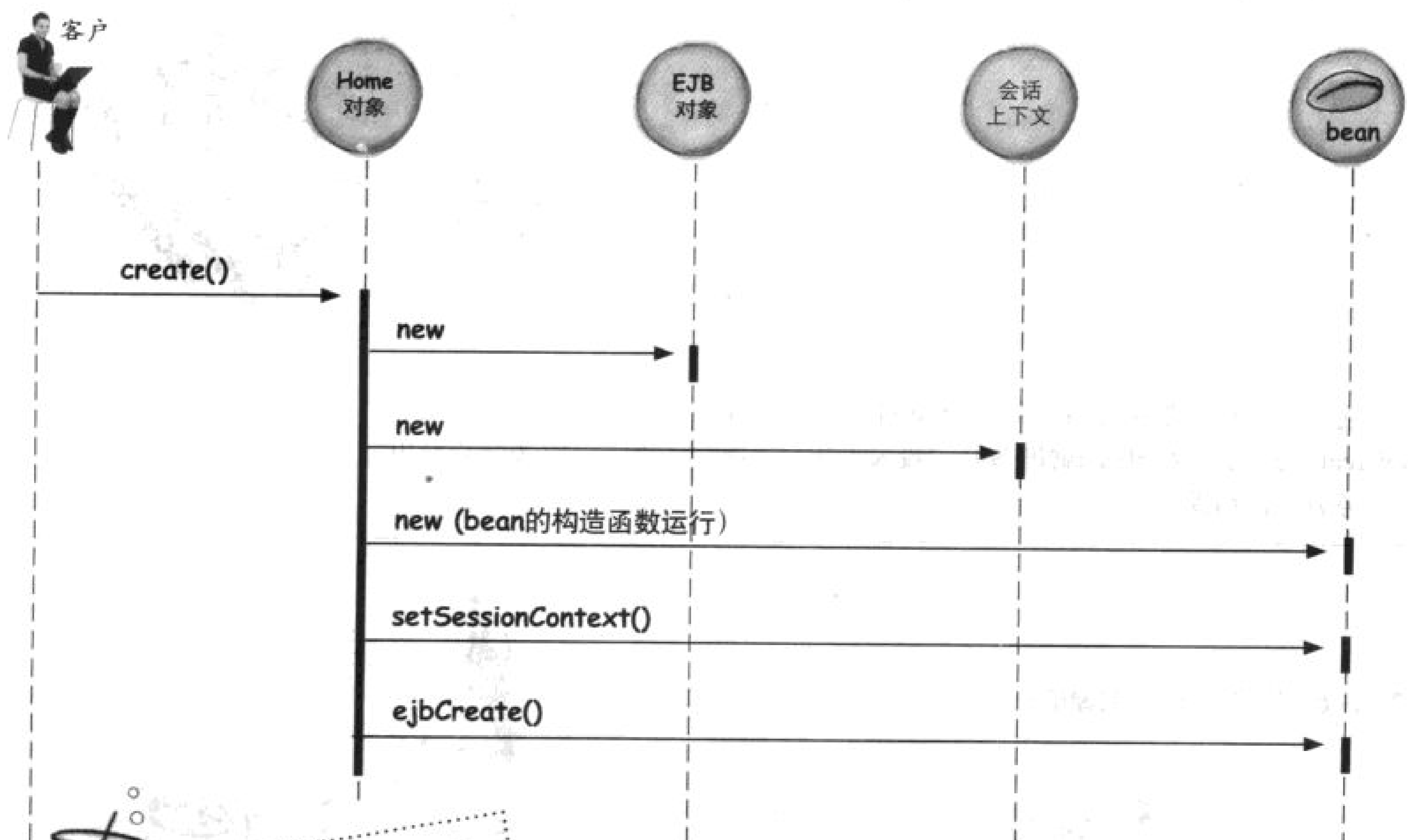
---

⑤ 容器向客户发送EJB对象的一个桩。



## bean创建的对象交互图（OID）

(有状态会话bean)







**要通过考试，无需了解UML！**

考试并不要求你通晓UML，尽管EJB规范中有一点UML。我们的一些最好的朋友就不懂UML，这没什么好惭愧的。真地。不过，如果最后只剩下10个不懂UML的人，而你不想成为异类被人看不起，可以考虑看看《Head First UML & Patterns》这本书，没准我们将来真会写这样一本书。

# 编写三个与创建有关的方法

(有状态会话bean)

## 1 构造函数

**不要在构造函数里做任何事情!** 能在构造函数里做的事情完全都能放在ejbCreate()里来做，所以，除非你的IDE为你放入了一个构造函数，否则最好不要在构造函数中写任何代码。

```
public class AdviceBean implements SessionBean {

    public AdviceBean() {
      // 这里没有任何代码!!
    }
  ...
}
```

构造函数就应该是这个样子！完全是空的。而且更好的做法是，根本不放入构造函数，而是让编译器给你一个默认构造函数。

## 2 setSessionContext(SessionContext sc)

**要保存你的上下文！** 你只有一次机会抓住SessionContext的这个引用，所以最好把它赋给一个实例变量。

```
private SessionContext context;

public void setSessionContext(SessionContext ctx) {
    context = ctx;
}
```

把SessionContext赋给一个实例变量保存起来。再没有机会来得到这个上下文的另一个引用了，所以最好在这里保存起来！要想在这个方法里再放其他的东西，可能把它们放在ejbCreate()里更合适。

## 3 ejbCreate()

**把所有初始化代码都放在这里！** 如果这是一个有状态bean，而且create()方法有参数，容器会把这些参数传给与创建方法匹配的ejbCreate()方法。现在已经是完整的bean了，所以从这个方法可以做任何需要做的事情，这包括在setSessionContext()中不能做的一些工作（如得到你自己的EJB对象的一个引用）。

```
private String name;

public void ejbCreate(String clientName) {
    name = clientName;
    // 其他代码!
}
```

你现在已经成为bean社会的一员了，所以通常会在这里放上所有初始化代码。



# 创建时能做的Bean工作：

(有状态会话bean)

时间 →

### 构造函数

无

现在太早了，还不能做任何事情！你现在只是一个对象，而不是一个真正的bean。

### setSessionContext()

使用你的SessionContext可以做：

- [x] 得到home的一个引用
- [ ] 得到EJB对象的一个引用
- [ ] 得到有关客户的安全信息
- [ ] 要求一个事务回滚 (CMT bean)
- [ ] 确定事务是否已经置为回滚 (CMT bean)
- [ ] 得到一个事务引用，并在上面调用方法 (BMT bean)

可以访问：

- [x] 你的特殊JNDI环境
- [ ] 另一个bean的方法
- [ ] 资源管理器（如数据库）

### ejbCreate()

使用你的SessionContext可以做：

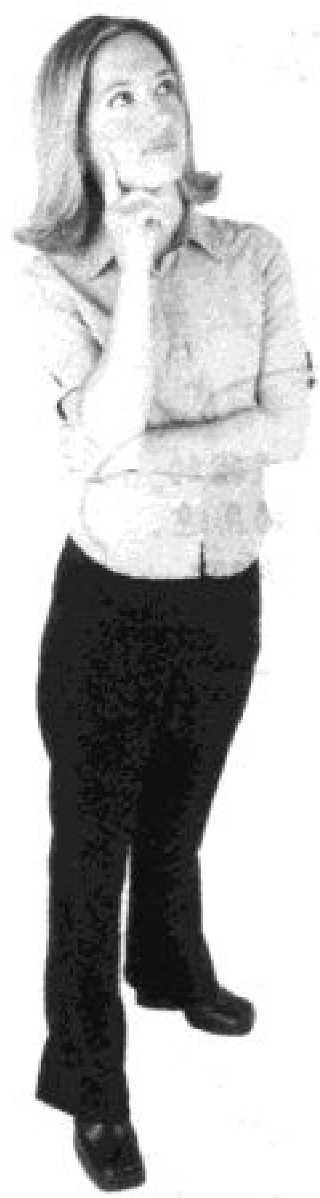
- [x] 得到home的一个引用
- [x] 得到EJB对象的一个引用
- [x] 得到有关客户的安全信息
- [ ] 要求一个事务回滚 (CMT bean)
- [ ] 确定事务是否已经置为回滚 (CMT bean)
- [x] 得到一个事务引用，并在上面调用方法 (BMT bean)

可以访问：

- [x] 你的特殊JNDI环境
- [x] 另一个bean的方法
- [x] 资源管理器（如数据库）

**注意**："访问（access）"这个词的意思是"用……做……"，所以规范中如果说"资源管理器访问"，就是说"使用资源来做事情，实现这个资源本来的用途"。例如，在setSessionContext()中，可以使用JNDI来查找一个资源管理器连接工厂，如javax.sql.DataSource，而且可以向这个DataSource要一个连接(myDataSource.getConnection())，但是不能在连接引用上做JDBC调用。不过在ejbCreate()中可以真正使用这个连接。

嗯，知道这些很好，不过这与我有什么关系？这对我来说意味着什么？它会对我的代码有什么影响？

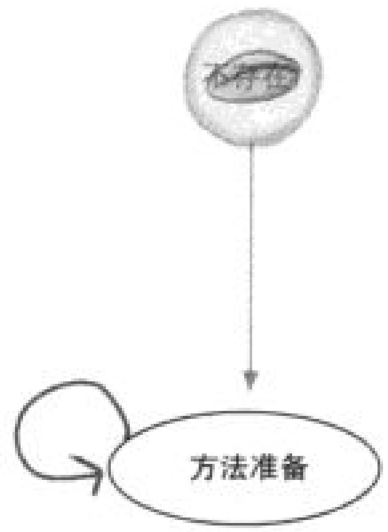


勤学好问的bean提供者

她还没完全搞明白……首先，我不会在构造函数里放任何东西！有一次我曾经试图在一个构造函数里做JNDI查找，想要得到一个数据库资源工厂，编译是通过了，在查找的时候也没有抛出异常，不过到了运行时，当我想得到一个连接时，它就出问题了！bean此时还没有拥有它的特殊bean JNDI功能，所以那里还没有资源工厂。想在构造函数里做的事情都能在ejbCreate()方法中完成。最好是把ejbCreate()想成是会话bean的构造函数（实体bean又是另一回事）。

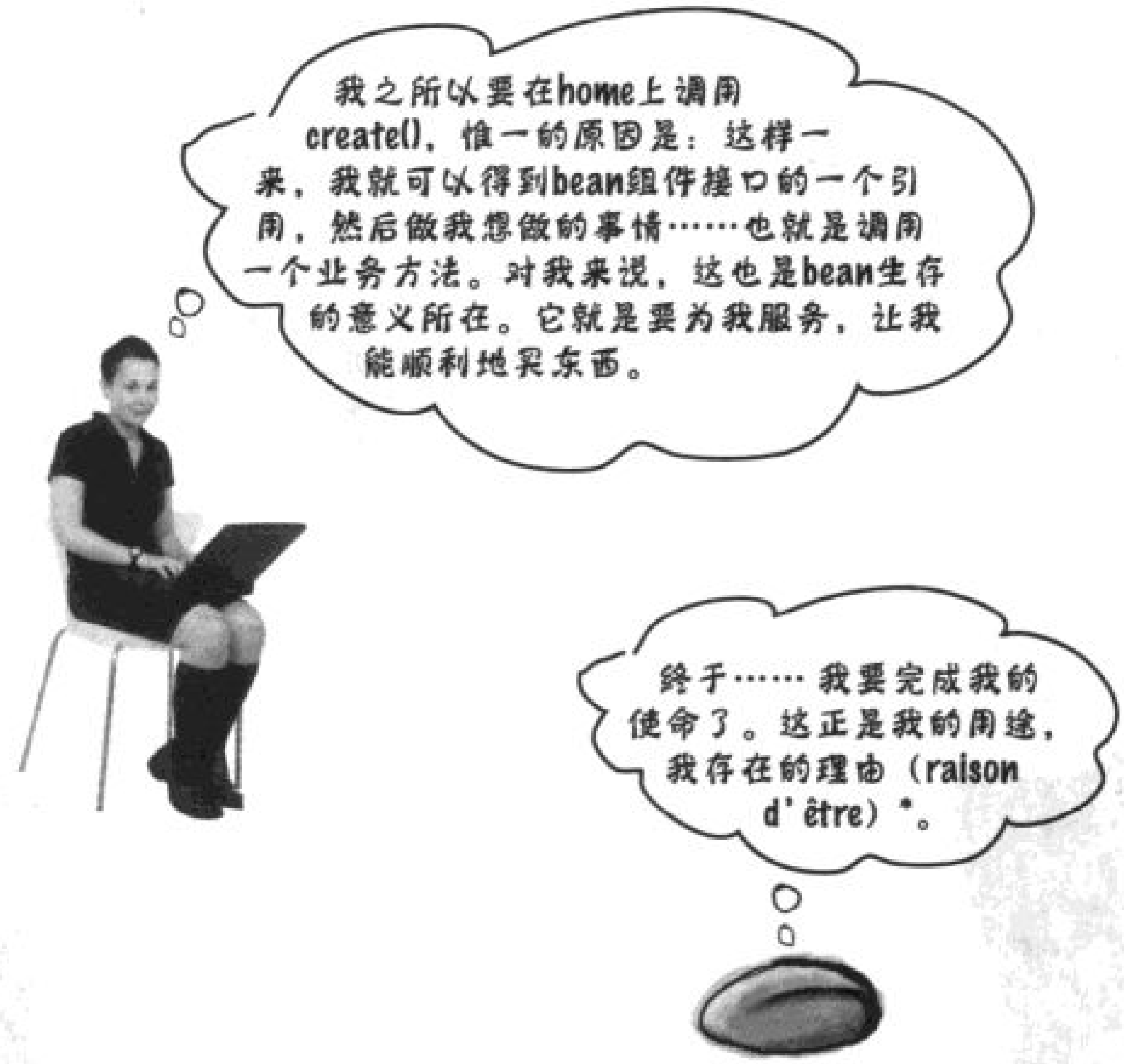


经验更丰富的bean提供者




在方法准备状态，bean要么在运行一个业务方法，要么在等待客户来调用一个业务方法……

# bean使用：创建之后会发生什么事情……




在方法准备状态，bean可以运行业务方法。这些业务方法正是bean的最终用途，此时bean可以把它的所有bean性运用到极致。换句话说，在bean简短但有意义的一生中，与其他时刻相比，在业务方法中bean能做更多的bean工作。

*我们不相信你还能找到另外一本计机书能像我们这么贴切地使用“存在的理由”（raison d'être）一词，还有ê上面的那个小尖角（倒v）等等。

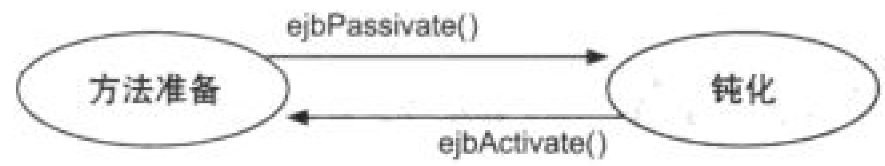
## 在业务方法中能做的bean工作：

**使用你的SessionContext可以做：**

- ☑ 得到home的一个引用
- ☑ 得到EJB对象的一个引用
- ☑ 得到有关客户的安全信息
- ☑ 要求一个事务回滚 (CMT bean)
- ☑ 确定事务是否已经置为回滚 (CMT bean)
- ☑ 得到一个事务引用，并在上面调用方法 (BMT bean)

**可以访问：**

- ☑ 你的特殊JNDI环境
- ☑ 另一个bean的方法
- ☑ 资源管理器（如数据库）




# 钝化：有状态bean实现可扩展性的机会……




要记住，有状态bean直接与客户关联，所以直到容器决定让bean超时之前（因为客户太长时间没有动作），不能用bean做任何其他的事情（或者为其他的客户服务）。

无状态bean没有这个问题，在每个方法调用的间歇，它们都会回到一个池中，因为无状态bean没有与客户绑在一起。

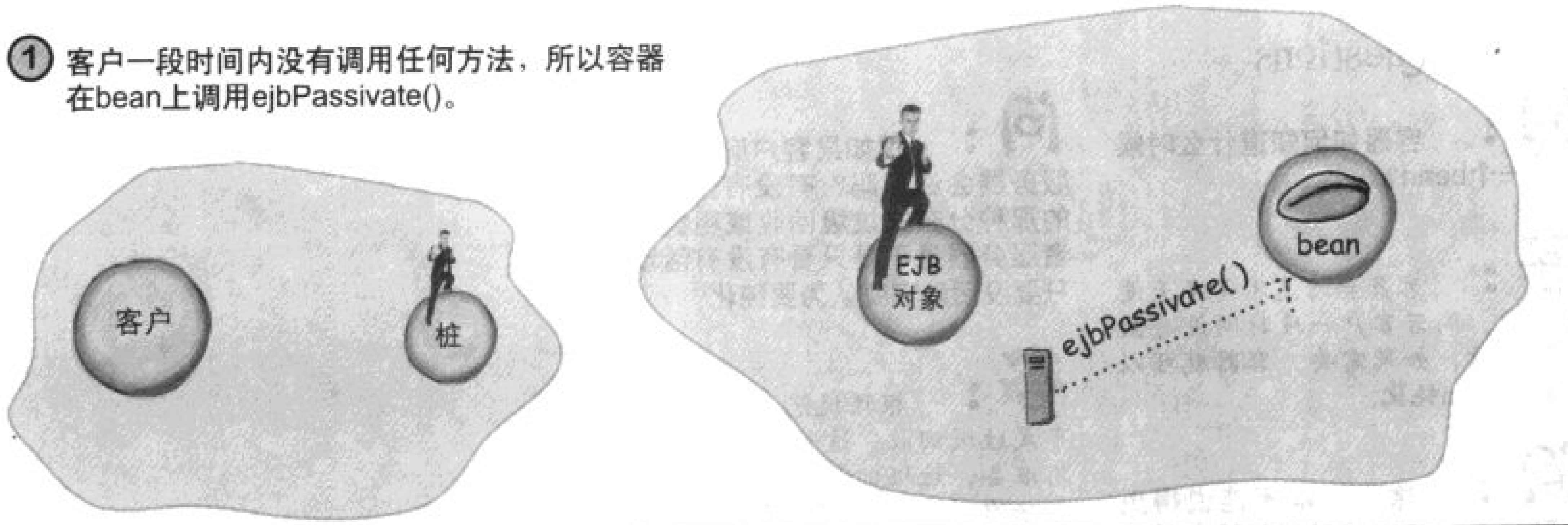
容器认为一个有状态bean在浪费资源时，它会调用这个bean的ejbPassivate()方法，然后把bean保存到临时存储区。

### 为什么要关心钝化细节？

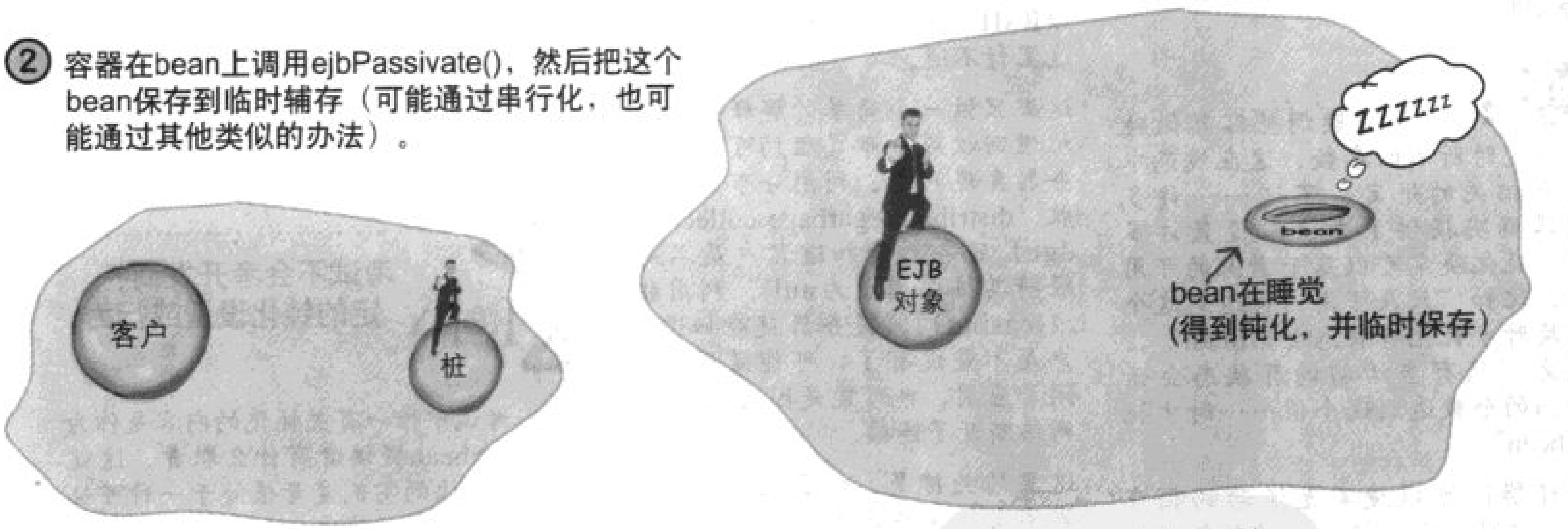
因为容器使用一组特殊的规则（与串行化几乎相同）来钝化bean，要确保你的实例变量处在一种可以钝化的状态下，这是你的任务。而且，相对于简单地将不可串行化的值置为临时（transient），这里只会稍有一点麻烦。

## 生命周期概述：bean钝化/激活

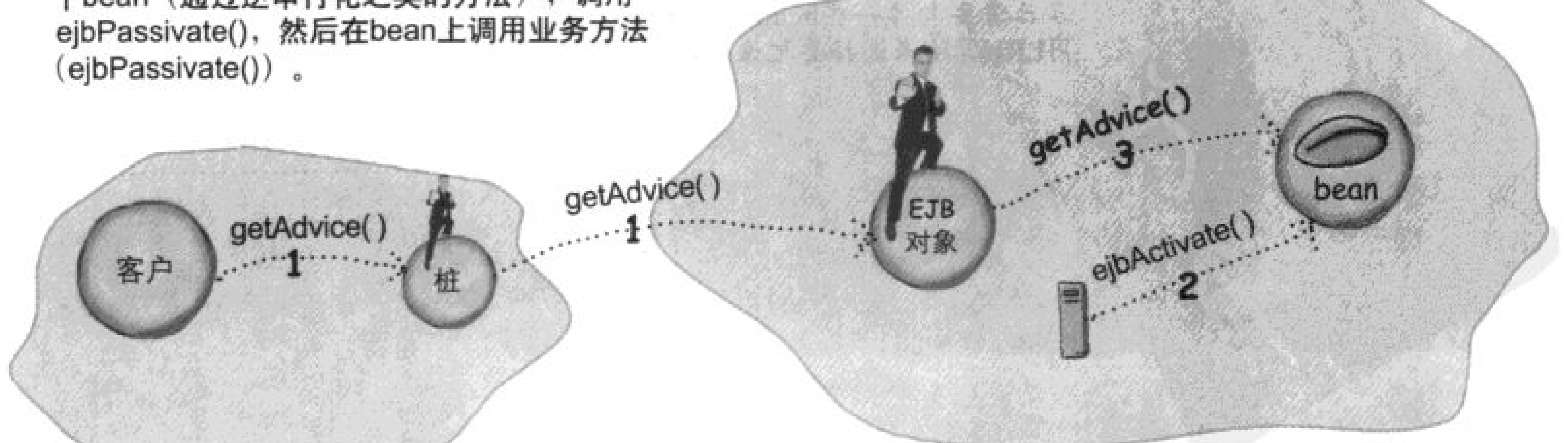
① 客户一段时间内没有调用任何方法，所以容器在bean上调用ejbPassivate()。




② 容器在bean上调用ejbPassivate()，然后把这个bean保存到临时辅存（可能通过串行化，也可能通过其他类似的办法）。




③ 客户调用了一个业务方法，所以容器激活这个bean（通过逆串行化之类的方法），调用ejbPassivate()，然后在bean上调用业务方法（ejbPassivate()）。



## there are no Dumb Questions

**问：容器如何知道什么时候钝化一个bean？**

答：容器会看是不是半天没有活动。倘若客户一段时间没有做任何事情，如果需要，容器就可以选择将bean钝化。

**问：喂……能不能说得更明确一些？“一段时间”是多长？怎么来确定容器是否“需要”钝化bean？**

答：是否能调整容器做钝化决定时所用的参数，这在规范中没有相关的规定。不过……许多容器确实提供了办法来设置许多值。钝化决定可以基于资源的可用性，比如“只在可用内存达到这个限度时才钝化有状态会话bean”，或者“只有当活动的有状态会话bean的个数达到这个值……时才钝化bean”。

你可能还可以设置无活动的相应值，也就是没有任何活动的耗用时间达到多少时容器就应该钝化了。

**问：那如果客户崩溃了呢？服务器会通知吗？有没有像RMI里的那种分布式垃圾回收或租约？或者服务器是不是只看有没有活动，只要没活动它就认为要钝化？**

答：根据规范，不能保证分布式垃圾回收，这说明你要做好思想准备，垃圾回收可能没有发生。部分原因在于，EJB客户可以（至少从理论上讲可以）是一个CORBA客户，而不是一个Java RMI客户。所以RMI中分布式垃圾回收的概念在这里行不通。

这里只做一个简单的解释：分布式垃圾回收或分布式租约可以帮助服务器发现问题，利用分布式垃圾回收（distributed garbage collection，dgc）服务器能知道客户是不是已经将其桩引用置为null，利用租约（leasing），服务器就能知道客户是不是没有了，可能是因为关闭了应用，也可能是因为崩溃或与网络断开了连接。

这里的关键是，你要假设EJB服务器不支持这些方法，所以只能看是不是没有活动，可能还可以用某种LRU（Least Recently Used，最近最少使用）算法，这样在资源变少，容器需要拿掉一些bean时，就可以用LRU算法来选择要把谁钝化。

**Relax**

**考试不会考开发商特定的钝化设置或行为。**

考试中惟一有关钝化的内容是作为一个bean提供者有什么职责，这就是让你的实例变量值处于一种可钝化的状态（稍后会介绍这个内容）。任何有关钝化的内容都不会放到部署描述文件中，所以你不必知道怎样为任何特定服务器设置钝化参数。

# 有关钝化，你的任务是：使你的状态可以钝化！




**ejbPassivate()完成时，每个非临时实例变量都必须是以下某一样的一个引用：**

- 一个Serializable对象
- 一个null值
- bean远程组件接口或远程home接口的一个引用，即使桩类不是Serializable（换句话说，你不用担心桩类是不是可串行化的！）
- bean本地组件接口或本地home接口的一个引用，即使它不是Serializable（同样地，你不用担心它是不是可串行化的）
- 一个SessionContext对象，即使它不是Serializable
- bean的特殊JNDI上下文，或者它的任何子上下文
- UserTransaction接口（ 从SessionContext可以得到，将在关于事务的一章再做介绍）
- 一个资源管理器连接工厂（比如，javax.sql.DataSource的一个实例）

**考试要求你必须了解这个内容。**

可能会有这样一个题目，给你一个类和一个ejbPassivate()方法，让你看看它能不能工作。
如果方法为空，就必须查看实例变量，看它们是不是不做任何处理就能钝化。

# 实现ejbActivate()和ejbPassivate()

## ejbPassivate()

**要确保你的实例变量已经准备好钝化。** 大多数情况下，ejbPassivate()里没有任何代码，因为你的所有实例变量都满足上一页定义的规则（例如，可能是一个SessionContext的引用，或bean组件接口的一个引用，或者一个Serializable对象，等等）。

```
public void ejbPassivate() {
   connection = null;
}
```

JDBC连接不是Serializable，所以如果有这样一个实例变量，就必须在ejbPassivate()中把它置为null。

## ejbActivate()

**再次获得你的不可串行化资源，或者尽力恢复你的状态以备使用。** 如果正在运行ejbActivate()，这只能有一个原因：就是客户调用了一个业务方法。所以，在你的ejbActivate()方法中，必须确保已经准备就绪，可以应对将要发生的业务方法调用。这个方法很有可能为空，原因与ejbPassivate()相同。不过，如果你确实将一个不可串行化的引用置为null，就要在ejbActivate()中将其恢复。

```
public void ejbActivate() {
   try {
    connection = myDataSrc.getConnection();
   } catch (Exception ex) {...}
}
```

现在必须再把JDBC连接恢复回来。注意，DataSource并不一定要恢复，原因是它名字在容器"批准列表"上（因为它是一个资源管理器连接工厂，而容器必须对其钝化）。

## there are no Dumb Questions

**问：难道不能把这些不可串行化的变量简单地置为临时吗？临时（transient）不就是做这个用的吗？**

答：是的，你可以这么做，因为容器钝化一个bean时必须遵循一些规则（尽管容器会对钝化方法有所选择，可能不使用串行化，而是使用其他某种方法来完成任务）。

不过……规范里有一个非常非常小的注释，这个注释指出，尽管一般情况下，钝化表现得像串行化一样，但是这也有一个例外。这个例外就是串行化必须把临时字段恢复成默认值，而钝化不能保证这一点！

这是什么意思？想想看。这说明，你不能依赖于临时（transient）将实例变量恢复成其默认值，所以激活之后，你可能会发现，标记为临时的变量里可能不是你希望的值，而是其他的任何一个值。

所以，你当然可以使用临时，而且这样能使钝化更高效，但是这也意味着，你要在ejbActivate()中自己重置这些临时的变量。

**问：如果我有一个不可串行化的值，而且确实要在钝化期间维护，该怎么办呢？如果我没有把它保存起来，怎么能知道在ejbActivate()中要它恢复成什么值呢？**

答：这是一个很经典的串行化问题，不光是钝化才有这个问题。通常采用的技巧是向这个不可串行化的对象询问它的所有重要状态，把这些状态保存在可串行化的实例变量里。然后，在ejbActivate()期间，使用你能保存的这些值来重新构造不可串行化的对象，使之与钝化前的对象相同。例如，假设你有一个Dog bean，它有一个Collar变量，而且Collar不是Serializable。在ejbPassivate()中，调用Collar的获取方法，获得一些重要的值（如getColor()、getSize()等等）。把这些值保存到bean中的实例变量里，然后在ejbActivate()中实例化一个新的Collar，调用这个新Collar对象的设置方法，并以你保存的Collar属性作为设置方法的参数。

## Sharpen your pencil

你要负责确保一点，这就是当ejbPassivate()完成时，你的实例变量必须是我们在前几页上所列的某种情况。不过现在不要看！看看你能不能自己得出下面哪种情况可以安全地钝化……

- ☐ java.net.Socket对象的引用
- ☐ javax.sql.DataSource的引用
- ☐ bean远程组件接口的引用
- ☐ bean JNDI上下文的引用
- ☐ java.sql.Connection的引用
- ☐ javax.ejb.SessionContext对象的引用
- ☐ 值为null的一个临时变量
- ☐ 值为null的非临时Serializable变量
- ☐ 值不为null的一个临时变量
- ☐ 值为null的一个非临时、不可串行化的变量
- ☐ 值不为null的一个非临时、不可串行化的变量*

*我们尽量不在一句话里用太多的否定（non），但这里实在没办法。




*警告:*

***有状态会话bean还在事务中时是绝对不能钝化的!***




如果你在一个有状态会话bean的某个方法中开始一个事务，而这个方法完成时没有结束事务，根据规范这是允许的。一般说来，这真的是一个很不好的事情。对于初学者，如果一个客户调用了开始事务的方法，你不能保证这个客户肯定会在某个时间调用结束这个事务的方法（提交或回滚）。不过更主要的原因是，事务的时间越长，你的服务器就越有可能被破坏（不管是哪一种类型的bean，都存在这个问题）。对于有状态会话bean，让一个事务一直打开，这意味着bean不会被钝化，不管客户多久没有对这个bean做任何调用，它也不会钝化。我们将在事务一章里更多地介绍这个内容，不过就目前而言，只要知道，事务中的有状态bean是不会钝化的！

# ejbActivate()和ejbPassivate()中可以做的bean工作

## ejbPassivate()

使用你的SessionContext可以做：

- [x] 得到home的一个引用
- [x] 得到EJB对象的一个引用
- [x] 得到有关客户的安全信息
- [ ] 要求一个事务回滚（CMT bean）
- [ ] 确定事务是否已经置为回滚（CMT bean）
- [x] 得到一个事务引用，并在上面调用方法（BMT bean）

可以访问：

- [x] 你的特殊JNDI环境
- [x] 另一个bean的方法
- [x] 资源管理器（如数据库）

## ejbActivate()

使用你的SessionContext可以做：

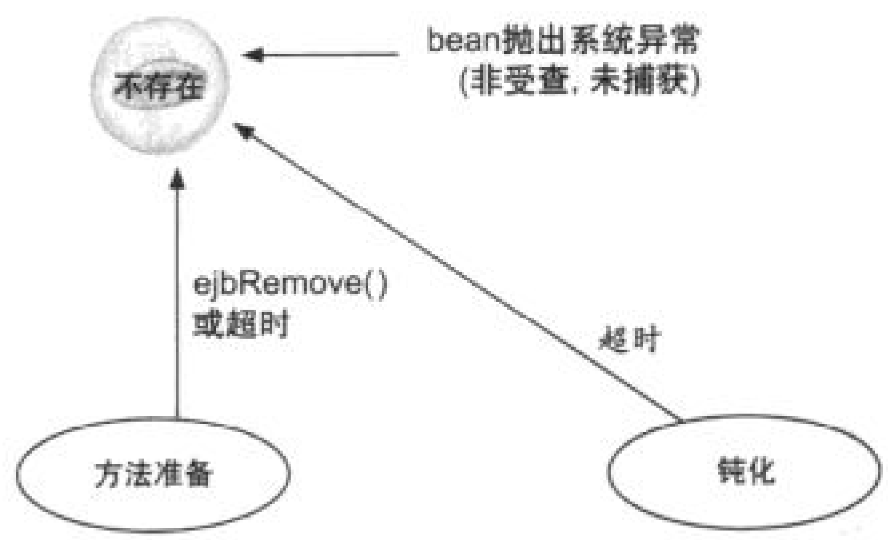
- [x] 得到home的一个引用
- [x] 得到EJB对象的一个引用
- [x] 得到有关客户的安全信息
- [ ] 要求一个事务回滚（CMT bean）
- [ ] 确定事务是否已经置为回滚（CMT bean）
- [x] 得到一个事务引用，并在上面调用方法（BMT bean）

可以访问：

- [x] 你的特殊JNDI环境
- [x] 另一个bean的方法
- [x] 资源管理器（如数据库）

# bean删除：bean什么时候死掉




## 会话bean不再存在，这有3个原因：

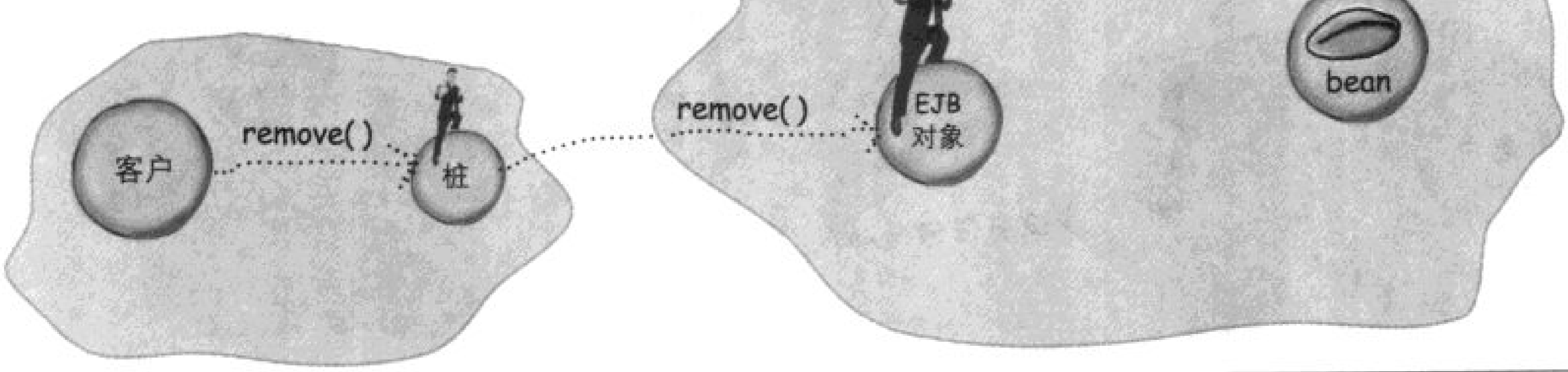
1. 客户调用了remove()
2. bean超时
3. bean抛出一个系统异常

不过……还有一个问题要问：bean超时的时候，它是活动的还是钝化的？如果这个bean是活动的，就像客户做了remove()调用一样，容器会做同样的处理，bean会得到一个ejbRemove()调用，并被杀死。但是如果bean超时的时候正处于钝化状态，容器就会直接把它送到不存在状态，而不会调用ejbRemove()。

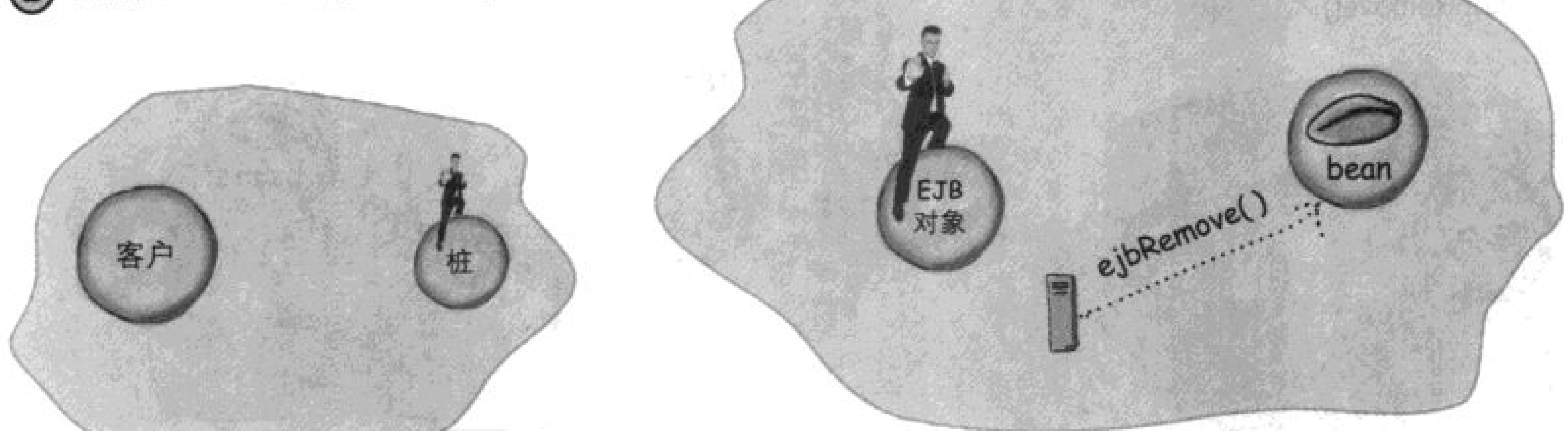
## 生命周期概述：删除一个有状态bean

客户在一个活动的（即非钝化）bean上调用remove()

① 客户在组件接口上调用删除方法（或者调用home接口中的remove()方法，这需要一个句柄）。




② 容器在bean上调用ejbRemove()。



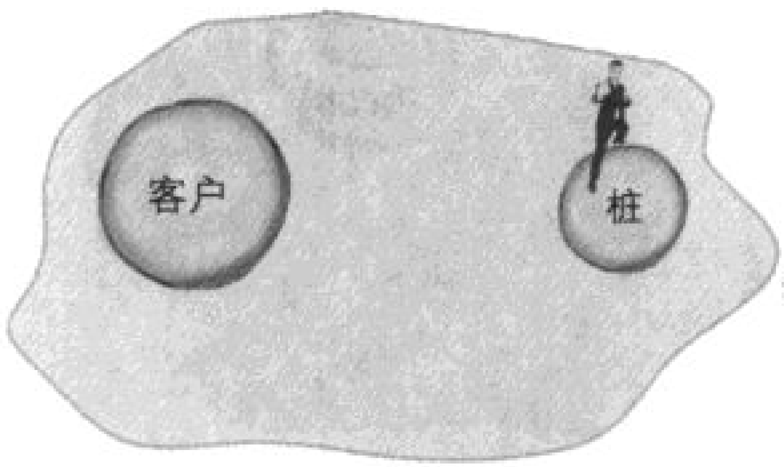

③ 容器“杀死”这个bean和EJB对象（假设它们现在都可以垃圾回收）。如果客户想再使用这个桩，她会得到一个异常。

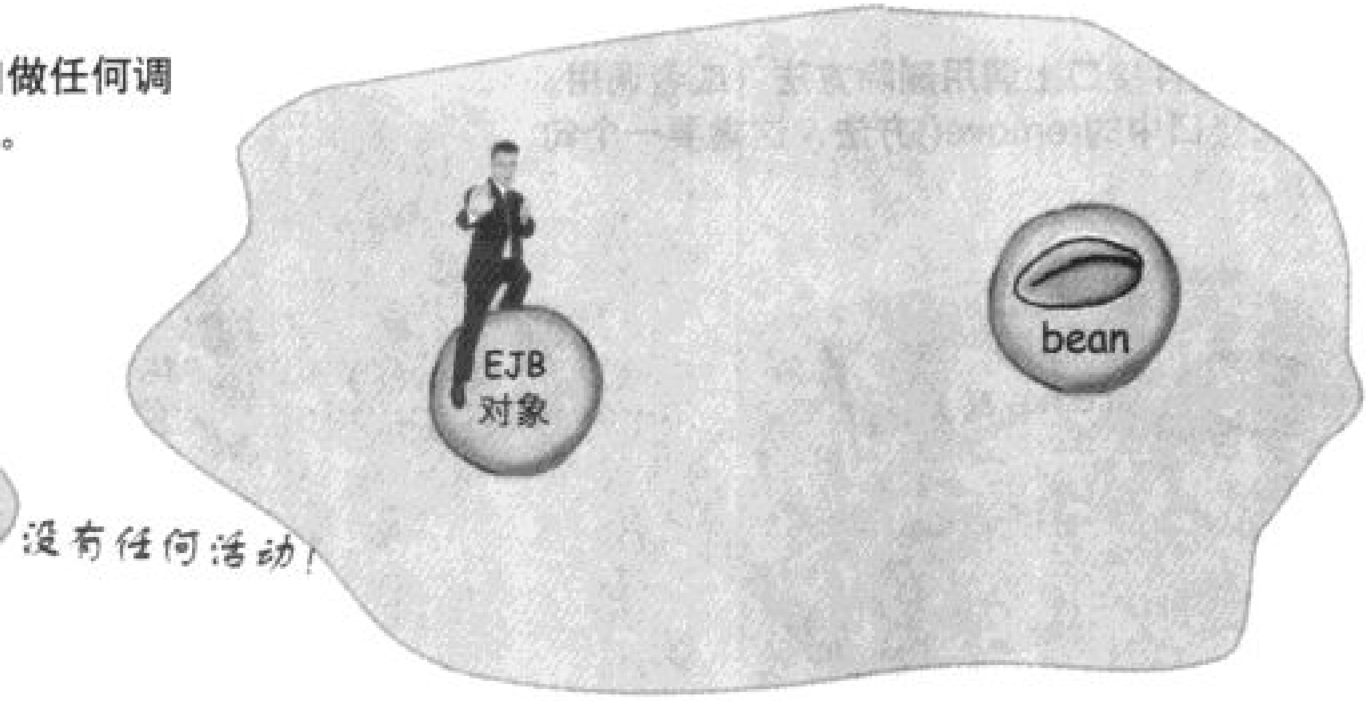


## 生命周期概述：删除一个有状态bean

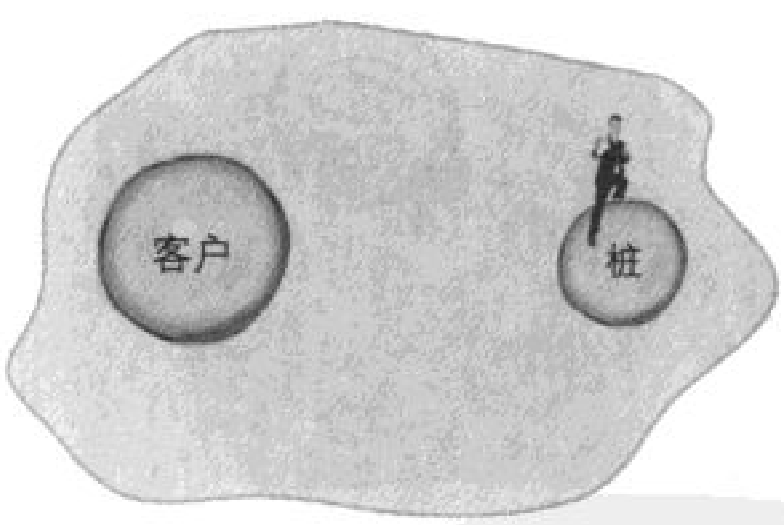
bean还是活动的情况下超时

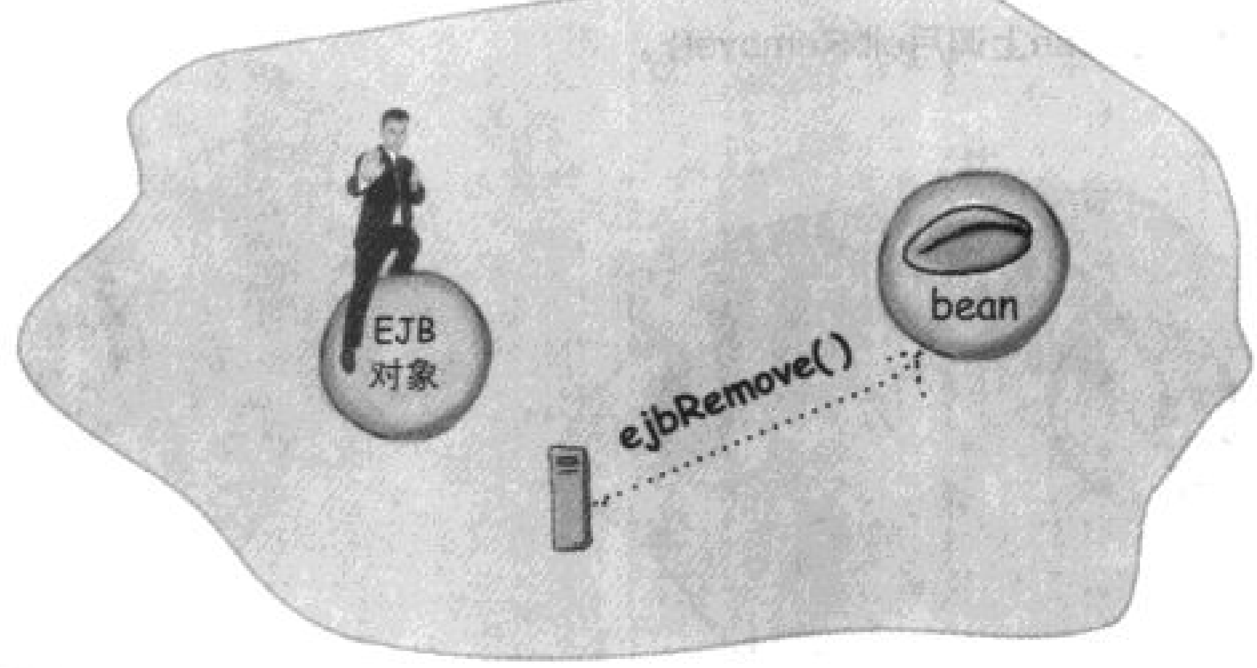
① 客户很长时间没有对bean的组件接口做任何调用（容器认为已经很“长”时间了）。



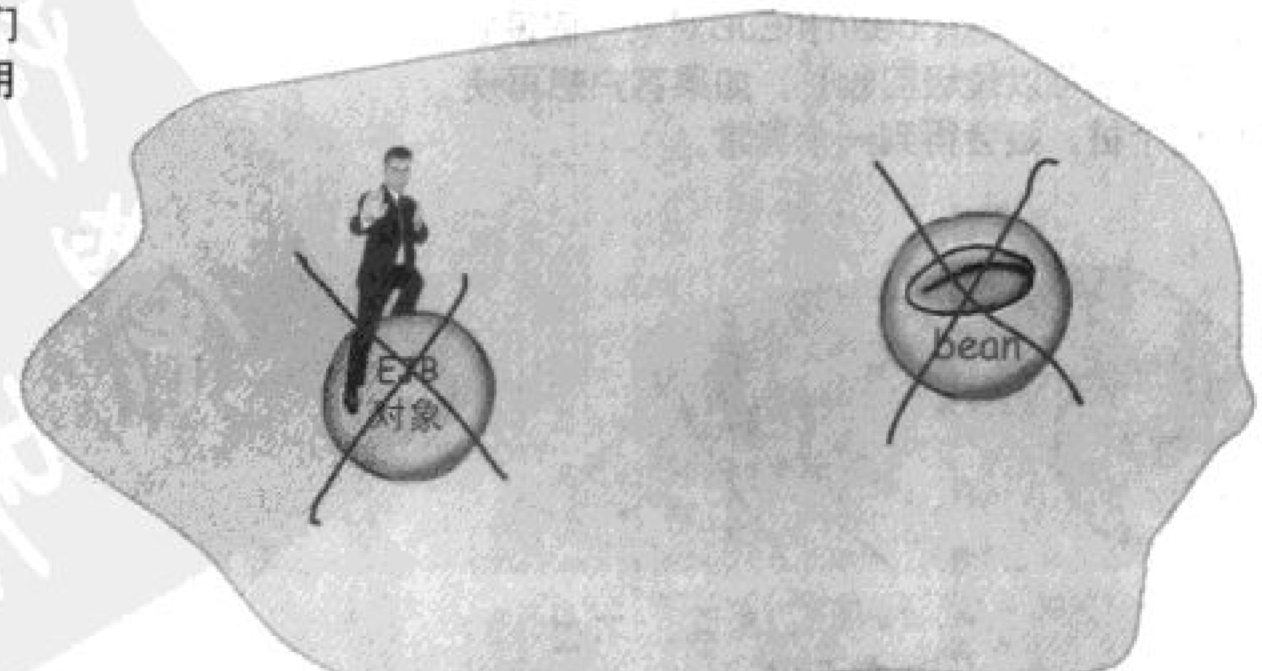

② 容器决定杀死这个bean，并在bean上调用ejbRemove()。




③ 容器“杀死”了bean和EJB对象（假设它们现在都能够垃圾回收）。如果客户想再使用这个桩，她会得到一个异常。

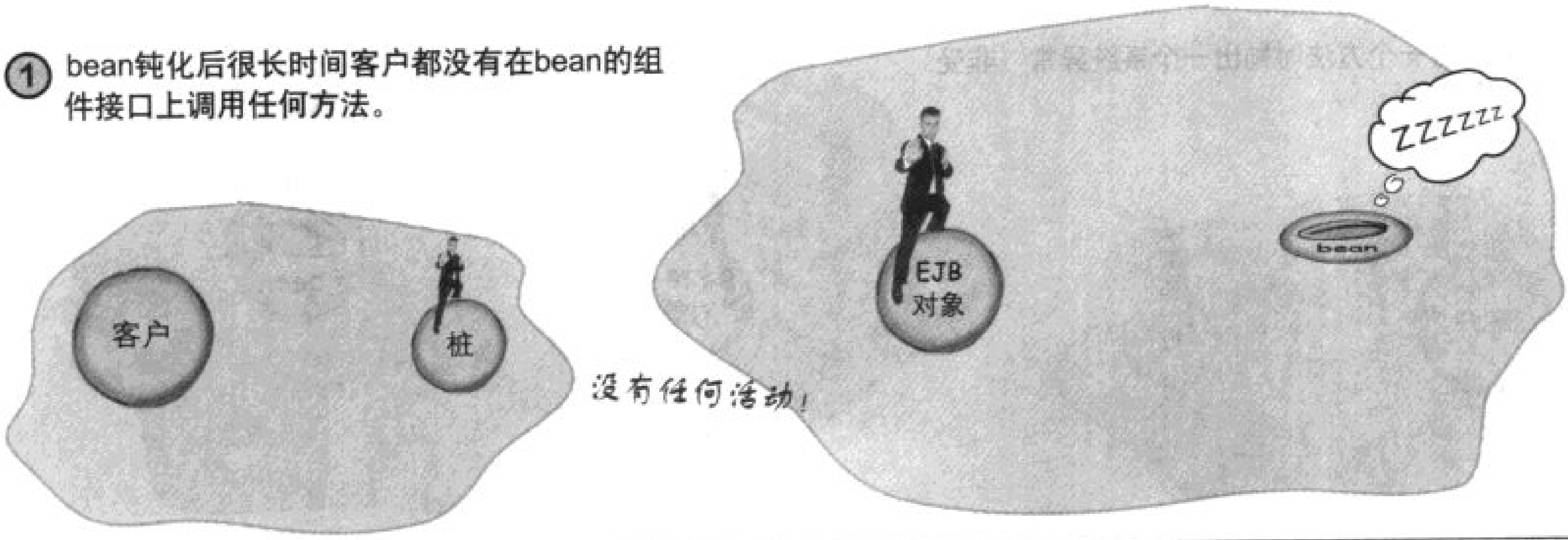


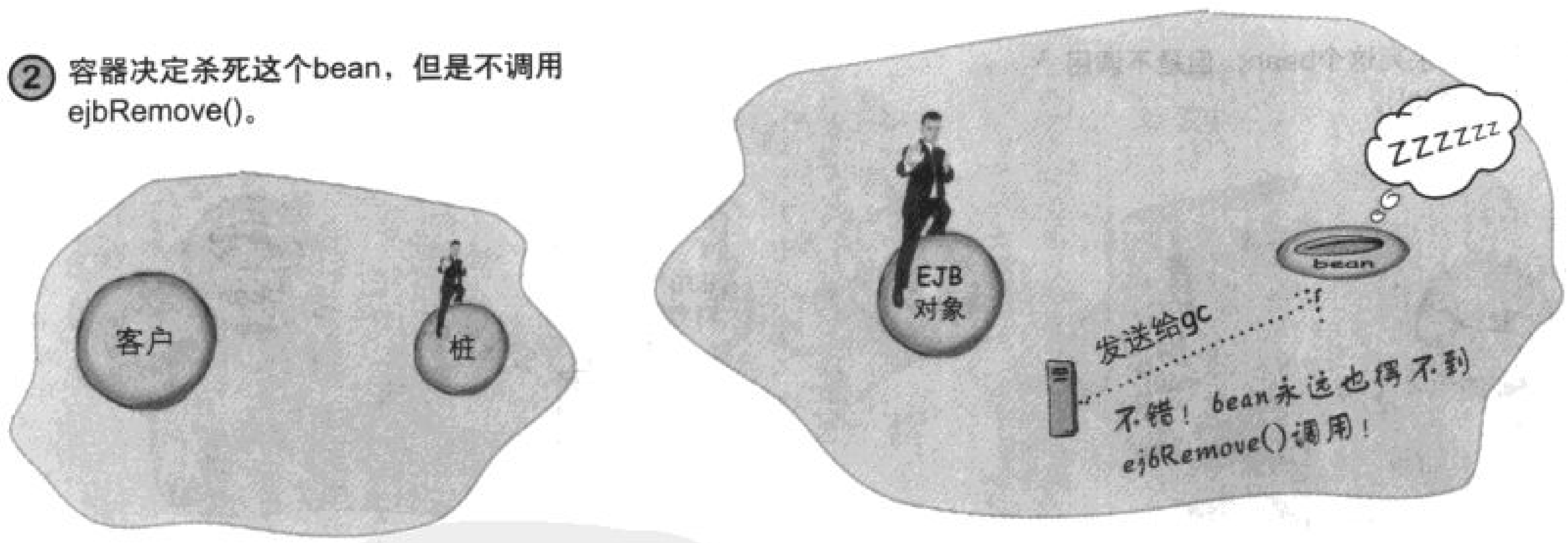
## 生命周期概述：删除一个有状态bean

bean钝化的情况下超时

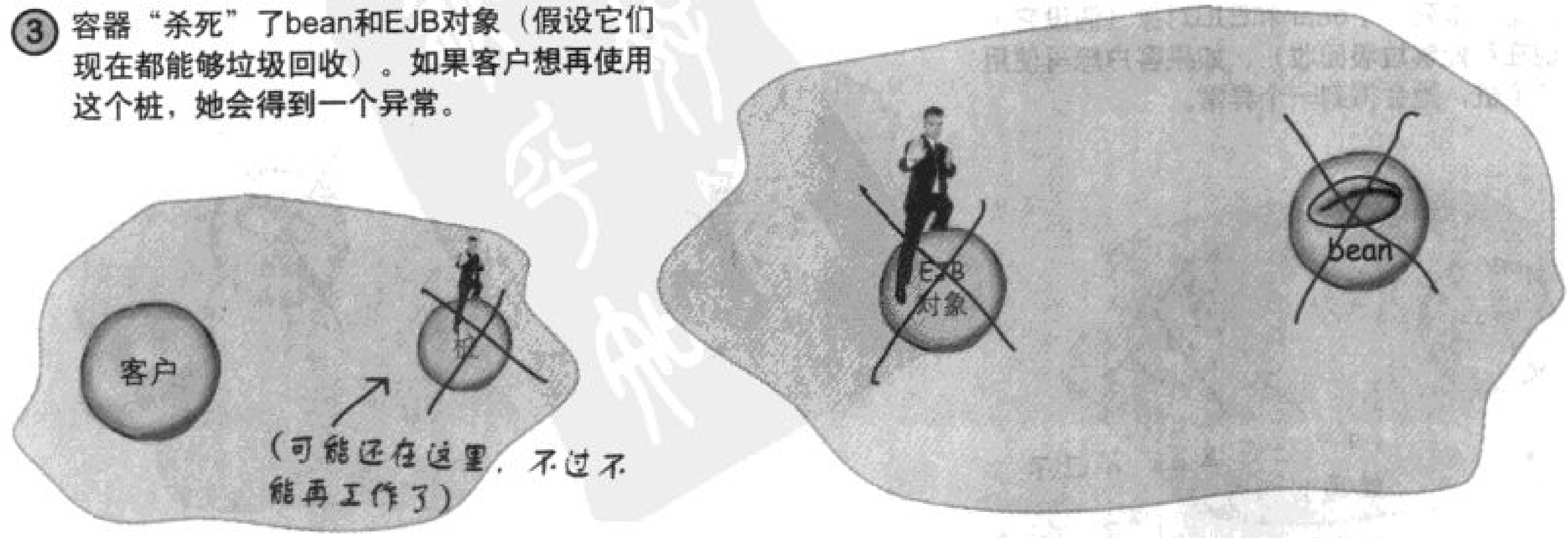
① bean钝化后很长时间客户都没有在bean的组件接口上调用任何方法。




② 容器决定杀死这个bean，但是不调用ejbRemove()。



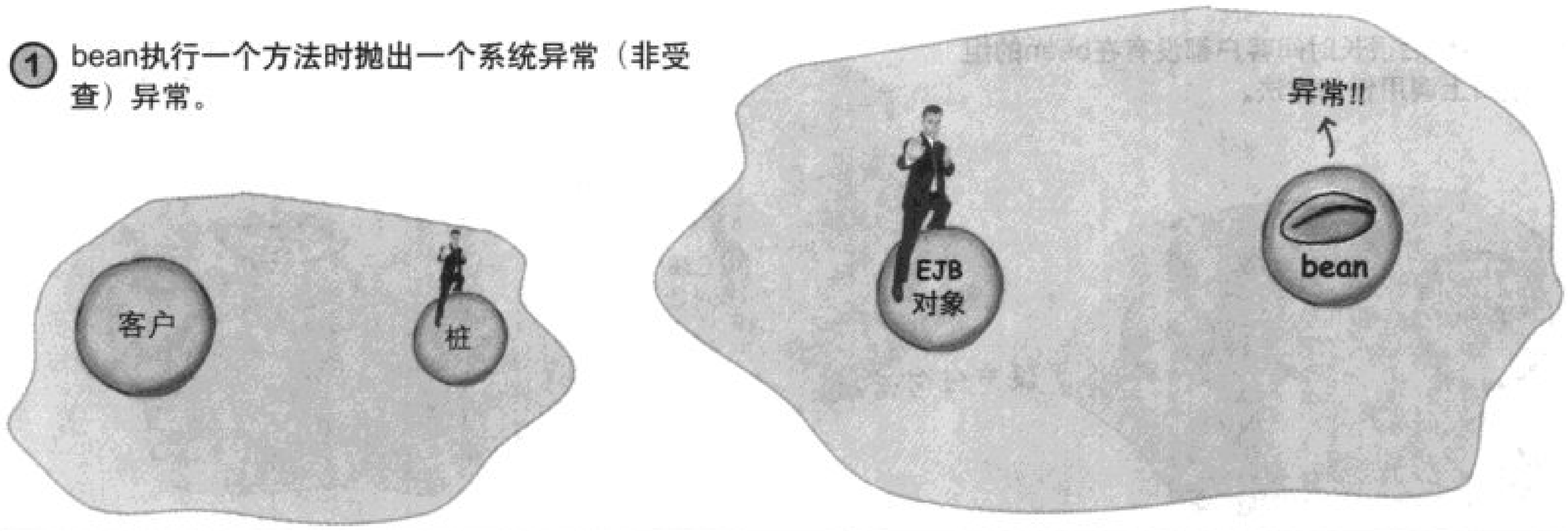

③ 容器“杀死”了bean和EJB对象（假设它们现在都能够垃圾回收）。如果客户想再使用这个桩，她会得到一个异常。



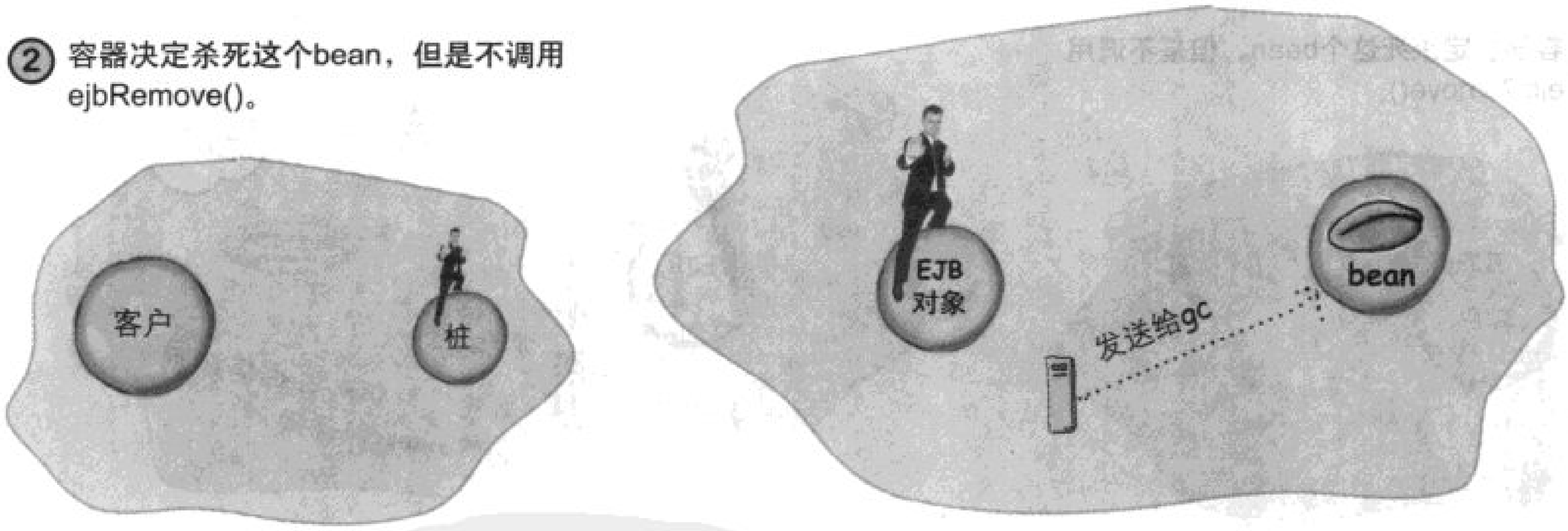
## 生命周期概述：删除一个有状态bean

bean抛出一个系统异常

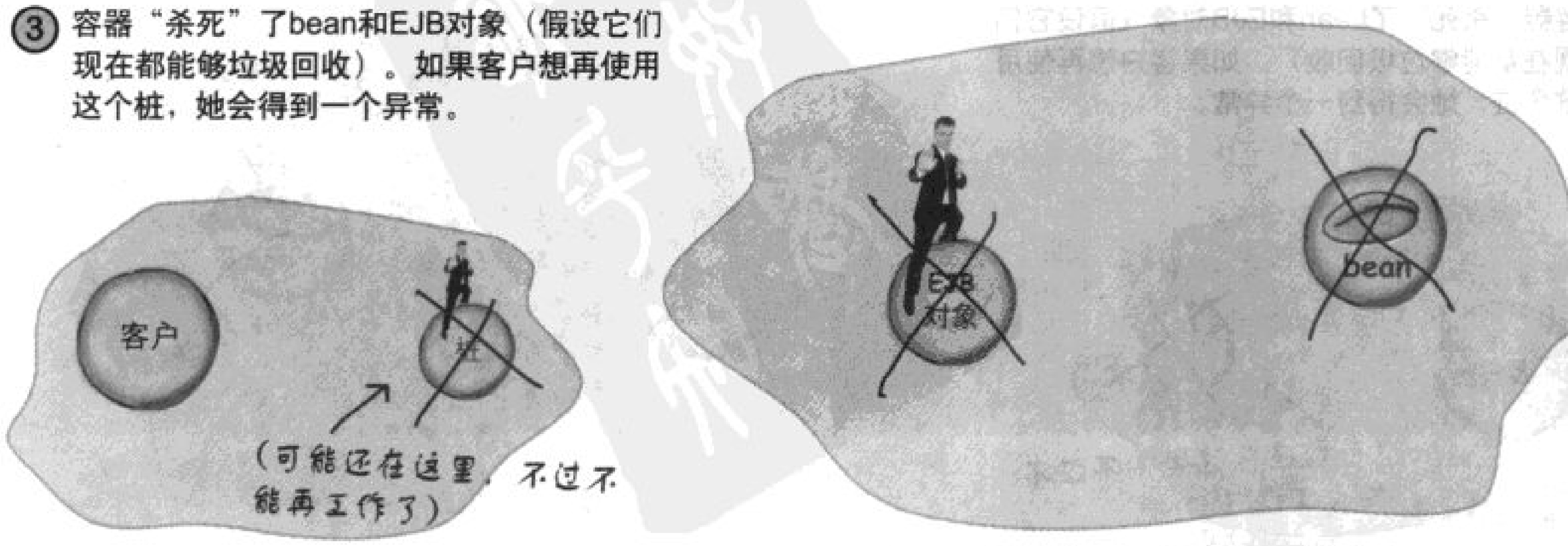
① bean执行一个方法时抛出一个系统异常（非受查）异常。




② 容器决定杀死这个bean，但是不调用ejbRemove()。




③ 容器“杀死”了bean和EJB对象（假设它们现在都能够垃圾回收）。如果客户想再使用这个桩，她会得到一个异常。



# 对bean删除的抱怨


为什么不等我调用remove()就把我的bean删掉？这样太粗鲁了！
我怎么知道你要回来？如果我不用超时的办法，不用过多久我就会被没有人要的破购物车塞满。我必须假设你永远也不回来，这是迟早的事。删了你的bean，对不起，不过，我真的别无选择。
我想不通，不管出于什么原因，既然你要删除bean，为什么不对它调用ejbRemove()呢？如果 ejbRemove()不会被调用，那在它里面放上清理代码又有什么用呢？
首先，你想让我把一个bean从钝化状态移出,是不是只是想杀死它？真是多此一举！如果想想性能，你就能清楚了。而且这不算大问题，完全可以在ejbRemove()和ejbPassivate()里都放上清理代码，这么做很安全。至于异常，这样来考虑……如果一个bean此时遭到严重破坏，你真的想在这个bean上运行清理代码吗？如果一个bean抛出一个系统异常，我只会把它记入日志，让它结束苦难。

# 实现ejbRemove()

## ejbRemove()

bean永远地死掉之前，要释放所有资源，或者完成所有需要的清理工作。在大多数情况下，ejbRemove()方法都是空的，因为如果你的bean确实使用了资源，很有可能是在各个业务方法里获得和释放这些资源。不过，如果你的设计确实需要让资源在bean的一生中一直打开，就需要在这里将其释放！

```
public void ejbRemove() {
  try {
     myResource.close();
  } catch(Exception ex) {...}
}
```

如果你想妥善地关闭还打开着的资源，就要在这里做这个工作！

## 另一种办法……

在ejbPassivate()和ejbRemove()中都调用一个cleanUp()方法。假设你的bean没有得到ejbRemove()调用也能被删除，例如它处在钝化状态的时候超时了，就应该让ejbPassivate()和ejbRemove()方法都完成同样的清理工作。由于系统资源往往都是可串行化的，所以很可能已经在ejbPassivate()中做了清理工作。

```
public void ejbRemove() {
   this.cleanUp();
}

public void ejbPassivate() {
   this.cleanUp();
}

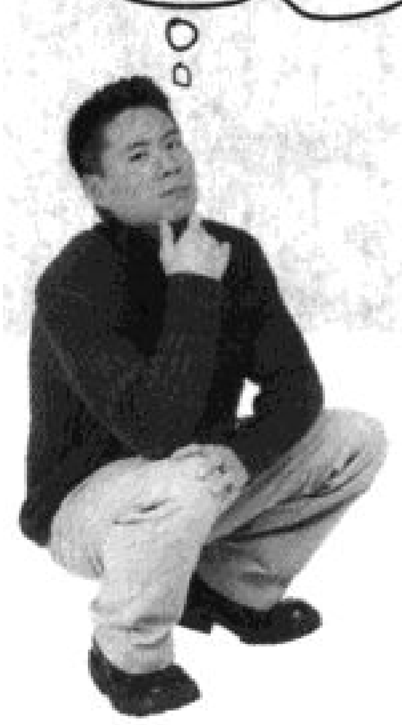
private void cleanUp() {
  try {
     myResource.close();
  } catch(Exception ex) {...}
}
```

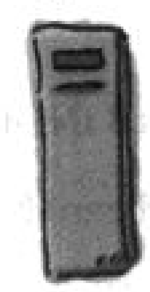
这样ejbRemove()和ejbPassivate()都会做同样的事情，所以即使你是在钝化时超时了，也不用担心（清理工作已经做了）。

# 没有得到ejbRemove()调用

请等等…… 这里还是有点问题。如果bean抛出一个异常会怎么样呢？这样我就不会运行清理代码。这不还是会有麻烦吗？

嗯，不错，如果是这种情况，你确实会有麻烦。但这是你的责任，要由你来处理。假设你把某人的购物信息临时保存在一个数据库里，这样如果这个人正在购物的时候服务器崩溃了，你还能有办法恢复他的购物车（直到bean超时）。在ejbRemove()中，你要在数据库中删除对应这个客户的记录行。但是由于你可能得不到这个ejbRemove()调用，必然需要一种办法来对这些数据完成周期性的清理。规范里对此没有任何说明。对于没有得到ejbRemove()调用可能带来的后果，要靠你自己来处理！





**当心！** 必须能看出来在哪些情况下会错过，没有得到ejbRemove()调用！

考试要求你知道，在3种情况下bean不会得到ejbRemove()调用：

(1) 服务器崩溃。
(2) bean在钝化时超时。
(3) bean抛出一个系统异常。

如果发生了上述某种情况，你的清理代码就不会运行。如果听取前一页的建议，在ejbPassivate()和 ejbRemove()上都放上清理代码，第2种情况（钝化时超时）就不会有问题，但是你还是要处理服务器崩溃或bean抛出系统异常的情况。

而且这方面的问题不会太简单，不会像“如果抛出一个系统异常，bean是不是会得不到ejbRemove()调用？”这么简单，这么明显。你必须考虑得更深，例如，可能必须考虑一个设计场景，根据你对得不到ejbRemove()方法调用的理解，看看这种情况下是否存在问题。



# ejbRemove()中可以做的bean工作

**使用你的SessionContext可以做：**

- [x] 得到home的一个引用
- [x] 得到EJB对象的一个引用
- [x] 得到有关客户的安全信息
- [ ] 要求一个事务回滚（CMT bean）
- [ ] 确定事务是否已经置为回滚（CMT bean）
- [x] 得到一个事务引用，并在上面调用方法（BMT bean）

**可以访问：**

- [x] 你的特殊JNDI环境
- [x] 另一个bean的方法
- [x] 资源管理器（如数据库）

ejbRemove()里能做的bean工作与以下方法中能做的bean工作完全一样：

- ejbCreate()
- ejbPassivate()
- ejbActivate()

# 把AdviceBean实现为一个有状态bean

到此为止，我们写的AdviceBean都没有必要是有状态的。它并不保存或使用客户特定的状态，所以不需要一个带参数的create()方法。但是，如果确实想让它成为一个有状态bean呢？如果业务逻辑需要保留以前所做会话的记录，这样就不会在一个会话中把同一条建议发出多次，该怎么办呢？即使建议串的选择是完全随机的，如果你想确保会话中的建议不会重复，可以在一个实例变量中把它记录下来。

还可以有其他办法，比如让创建方法取一些参数，参数中包含客户所查找建议的类型，或者是另外某种首选项。在这种情况下，客户每次做一个方法调用时，你就可以检查客户提供的创建初始化首选项的状态，并且在bean的ejbCreate()中根据这个值来调整你的建议。在这本书的后面，我们会看到这个AdviceBean的一个更复杂的版本，不过，对于现在来说，我们只对它稍做修改，让它变成一个有状态的bean。

## BRAIN POWER

你知道bean类要修改，那么客户和两个接口呢？是不是其中的哪一个或哪几个也需要修改呢？在翻到下一页之前，先好好考虑一下。

## 如果bean是有状态的，还可以：

1. **可以有多个创建方法。**

2. **创建方法可以有参数。**

3. **bean可以钝化，所以可以在ejbPassivate()和ejbActivate()中编写代码。**

### there are no Dumb Questions

**问：** **如果在ejbPassivate()或ejbActivate()中放上代码，而bean是无状态的，会怎么样呢？能编译吗？能部署吗？**

**答：** 当然可以在ejbActivate()或ejbPassivate()中放代码，而不管这个bean是无状态的还是有状态的，只不过如果bean是无状态的，这些方法绝不会被调用。

想想看。请看SessionBean接口。要注意，这不是一个单独的StatefulSessionBean接口和StatelessSessionBean接口。这里只有SessionBean，无状态和有状态bean都要实现这个接口。所以，你的类中没有什么会专门说你的bean是无状态的。只有在部署时，当你告诉部署描述文件这个bean是无状态的还是有状态的，才会有区别。

实际上，可以这样来编写bean，让它只有一个无参数的创建方法，而且不保留客户状态，这样不管如何设置（无状态或是有状态），它都能部署，而且都能工作，只要你正确地处理了钝化和激活。

# AdviceStatefulBean代码

```
package headfirst;
import javax.ejb.*;

public class AdviceStatefulBean implements SessionBean {

   private SessionContext context;

   private String userName;

   public void ejbActivate() {
   }

   public void ejbPassivate() {
   }

   public void ejbRemove() {
   }

   public void setSessionContext(SessionContext ctx) {
      context = ctx;
   }

   public void ejbCreate(String name) {
      userName = name;
   }

   public String getAdvice() {
      return userName + ", my advice is: " + Advisor.getAdvice();
   }
}
```

还是要实现SessionBean

这一次，可以保留客户特定的状态（在这里，就是客户发送给创建方法的参数）。

在4个SessionBean容器回调中，我们惟一要做的事情就是保存上下文，尽管在这个版本中并没有使用这个上下文。之所以要保存上下文，原因是我们希望你能养成保存上下文的习惯，因为在大多数实际bean中，你往往希望以后使用这个上下文（在本书的后面，你会看到使用上下文的例子）。

这只能是一个有状态bean，因为无状态bean除了无参数的create()方法外，不能有其他创建方法。

在这个业务方法中，我们使用了客户特定的状态，这是我们保存的创建方法参数。这必然是一个有状态bean，因为无状态bean不会与一个客户绑定。

假设这是Advisor类上的一个静态辅助方法（这里没有给出）。请你自己来写这个方法。

# AdviceStatefulBean客户代码

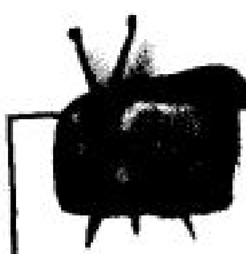
有状态版本的客户中有两项是无状态版本中所没有的：

*1. 向create()方法传递了一个参数。*

有状态bean的创建方法可以有参数（而且通常都是这样），这样客户就可以传入客户特定的状态，让bean保存起来，而且在以后与这个客户的“会话”（也就是说，以后这个客户所做的方法调用）中使用。

*2. 在组件接口上调用remove()。*

客户要告诉容器，她已经用完了一个有状态会话bean，这是一种礼貌。现在你知道这是为什么了。如果没有remove()调用，容器就会一直留着bean，直到bean超时，这是很浪费的。所以在这种情况下，“礼貌”的意思实际上就是“改善可扩展性”。



**必须知道客户是否对应一个有状态bean。**

你要会分析像这样的客户代码，知道客户的bean是不是有状态的。如果客户调用了一个无参数create()，从这一点并不能知道它是否为无状态bean，不过，至少可以知道它可能是无状态的。如果创建方法有参数，或者名字不是create()，而是其他（如createAccount()），就可以知道这个bean肯定是有状态的。

```
import javax.naming.*;
import javax.rmi.*;
import headfirst.*;
import javax.ejb.*;

public class AdviceStatefulClient {

    public static void main(String[] args) {
        new AdviceStatefulClient().go();
    }

   public void go() {
     try {
        Context ic = new InitialContext();
        Object o = ic.lookup("StatefulAdvisor");

        AdviceStatefulHome home = (AdviceStatefulHome) PortableRemoteObject.narrow(o,
                                                        AdviceStatefulHome.class);

        AdviceStateful advisor = home.create("clover");

        System.out.println(advisor.getAdvice());

        advisor.remove();
     } catch (Exception ex) {
          ex.printStackTrace();
     }
   }
}
```

这肯定是一个有状态bean，因为我们在调用有参数的创建方法！

我们在调用remove()，告诉容器已经用完了这个bean（这样容器就能把它删掉了）。

这种处理异常的方法很不好！可能要编写一个实际的客户来捕获另外一些类型的重要异常，如NamingException、CreateException等等。

## Sharpen your pencil

**从无状态变为有状态时，接口必须修改吗？**

请看下面的两个接口，它们对应无状态版本的Advice bean。如果需要，对这两个接口中的代码做所需的修改，从而能够用于这个bean的有状态版本。

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;

public interface AdviceHome extends EJBHome {
   public Advice create() throws CreateException, RemoteException;
}
```

```
package headfirst;

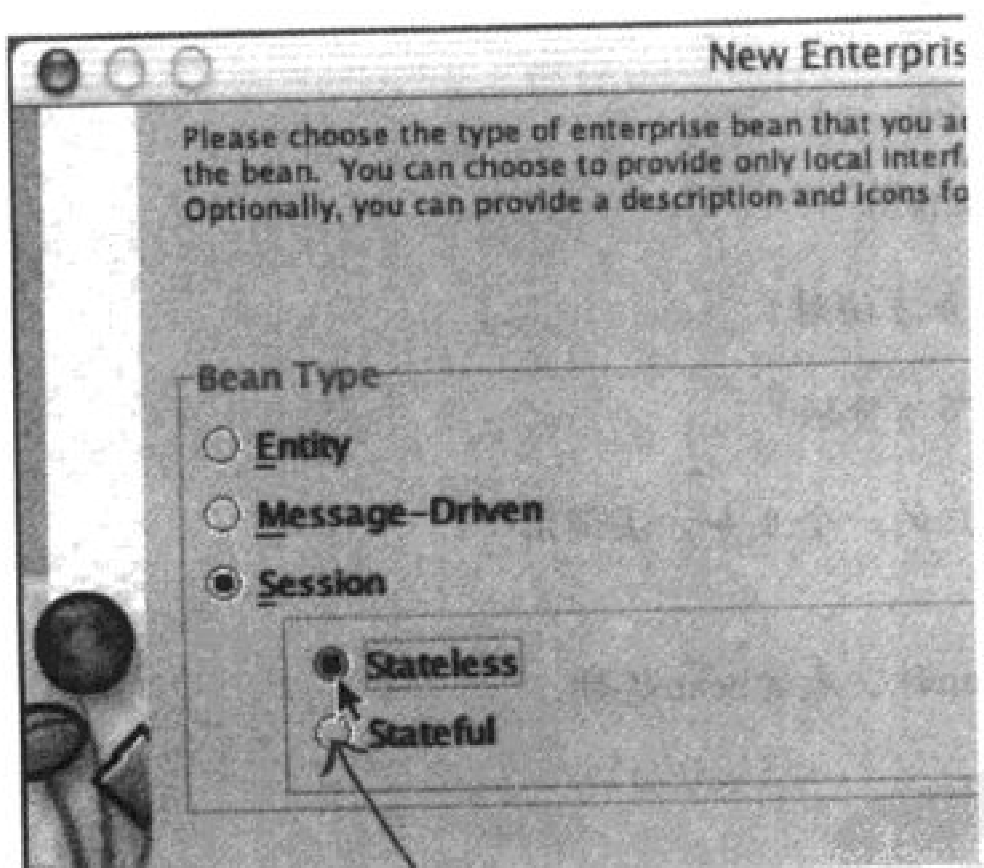
import javax.ejb.*;
import java.rmi.RemoteException;

public interface Advice extends EJBObject {
      public String getAdvice() throws RemoteException;

}
```

## 部署一个有状态bean

部署有状态Advice bean和无状态Advice bean只有一个区别（除了文件名和JNDI名），这就是部署描述文件设置中，把bean定义为无状态还是有状态。使用RI deploytool时，应该记得，你是使用deploytool的New Enterprise Bean向导工具来设置这一点的，这个向导工具带着你完成一系列设置，然后用这些设置输出一个XML部署描述文件。




你必须把它改为Stateful（有状态）!!

### 要点

- 容器回调指示了bean一生中的重要时刻。
- 作为一个bean提供者，你要负责在bean类中实现容器回调。
- 容器回调来自两个地方：SessionBean接口和home接口。编译器要求必须实现SessionBean接口，但是与home有关的回调方法就要由你自己负责了，编译器并不知道必须实现这些回调，因为你的bean类没有实现home接口。
- 有状态会话bean可以有3个状态：不存在（不错，这确实是一个状态）、方法准备和钝化。
- 一个bean从不存在状态变迁到方法准备状态时，会调用它的构造函数，然后调用setSessionContext()，最后调用这个bean的ejbCreate()。
- 一个bean实例有一些能做的特定bean工作，不过在bean的构造函数中什么也不能做，因为此时它还只是一个对象，而不是完全意义上的bean。它还不具备bean性。
- bean能做的一些bean工作包括：得到其home或EJB对象的一个引用、了解或影响事务的状态、得到有关客户的安全信息，以及访问诸如数据库等资源。
- 一个有状态bean钝化时，它会被放到辅存中，可能要通过串行化。你必须确保在ejbPassivate()方法的最后，你的实例变量已经准备好钝化。
- 对于一个有状态bean，客户调用remove()时，容器会在bean上调用ejbRemove()，如果是一个无状态bean，容器想缩小bean池的大小时，就会在bean上调用ejbRemove()。如果一个钝化的bean超时，容器就会直接杀死这个bean，而不用先调用ejbRemove()。
- 如果容器崩溃或者bean抛出一个运行时异常，bean还有可能得不到ejbRemove()调用。

# 与有状态bean相比，无状态bean的生活简单多了

无状态bean的人生要简单得多。首先它们要出生（创建），然后和其他同类的bean一起被扔到一个bean池中，如果客户有要求，它们就要为这些客户运行业务方法（不管是哪个客户），最后可能会死掉。无状态bean不会钝化，它们不保留客户特定的状态，而且它们的创建和撤销（删除）不操纵在客户的手里。请比较有状态和无状态bean的生命周期有什么区别：

## bean生命周期（bean一生中的特殊时刻）

### 有状态会话bean

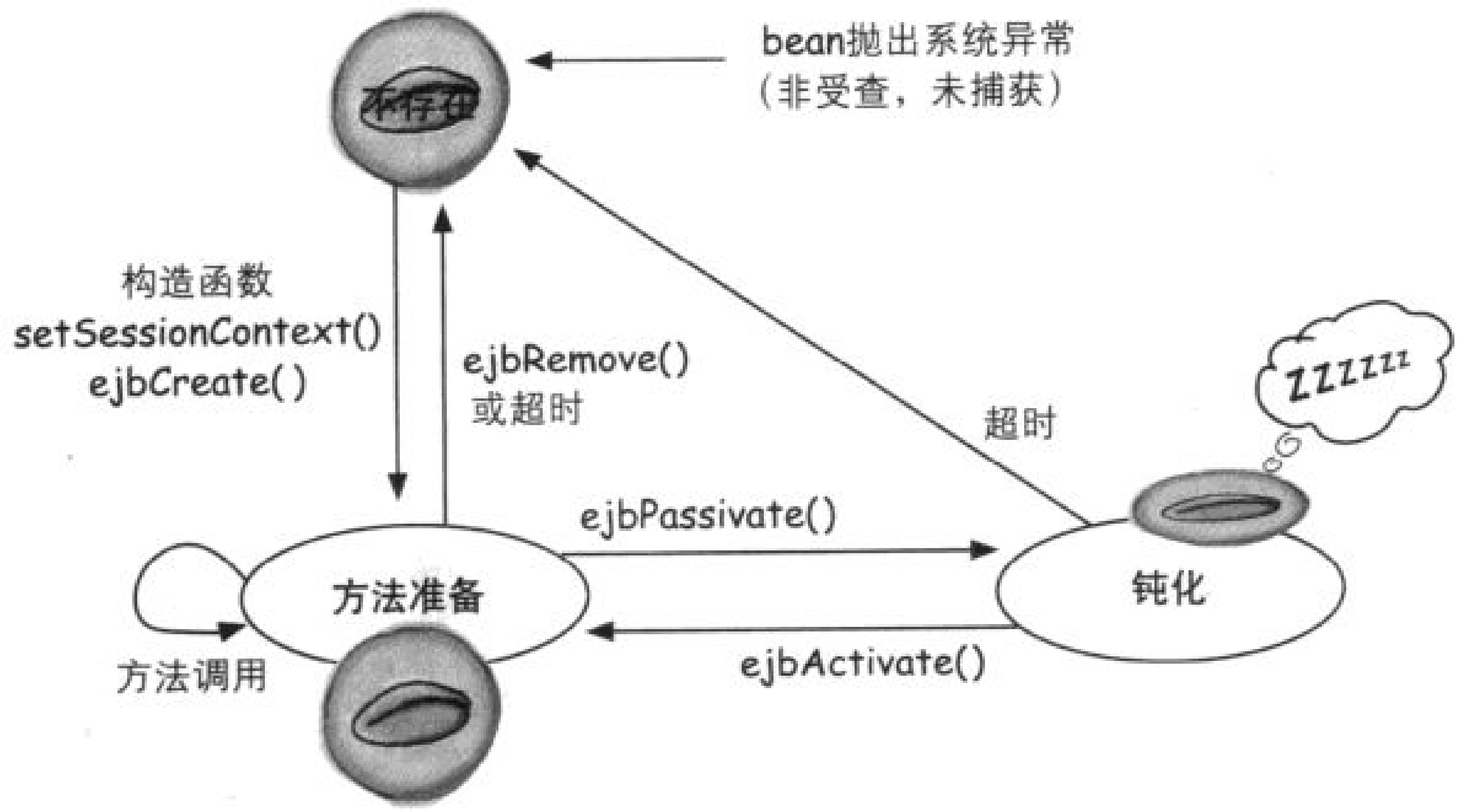
- **bean创建**（客户想要一个bean时）
- **bean使用**（客户调用一个业务方法时）
- **bean钝化**（让bean睡觉，来节省资源）
- **bean激活**（bean醒来，为客户的一个业务方法调用提供服务）
- **bean删除**（客户用完这个bean时，或者bean超时，或者 bean抛出系统异常）

### 无状态会话bean

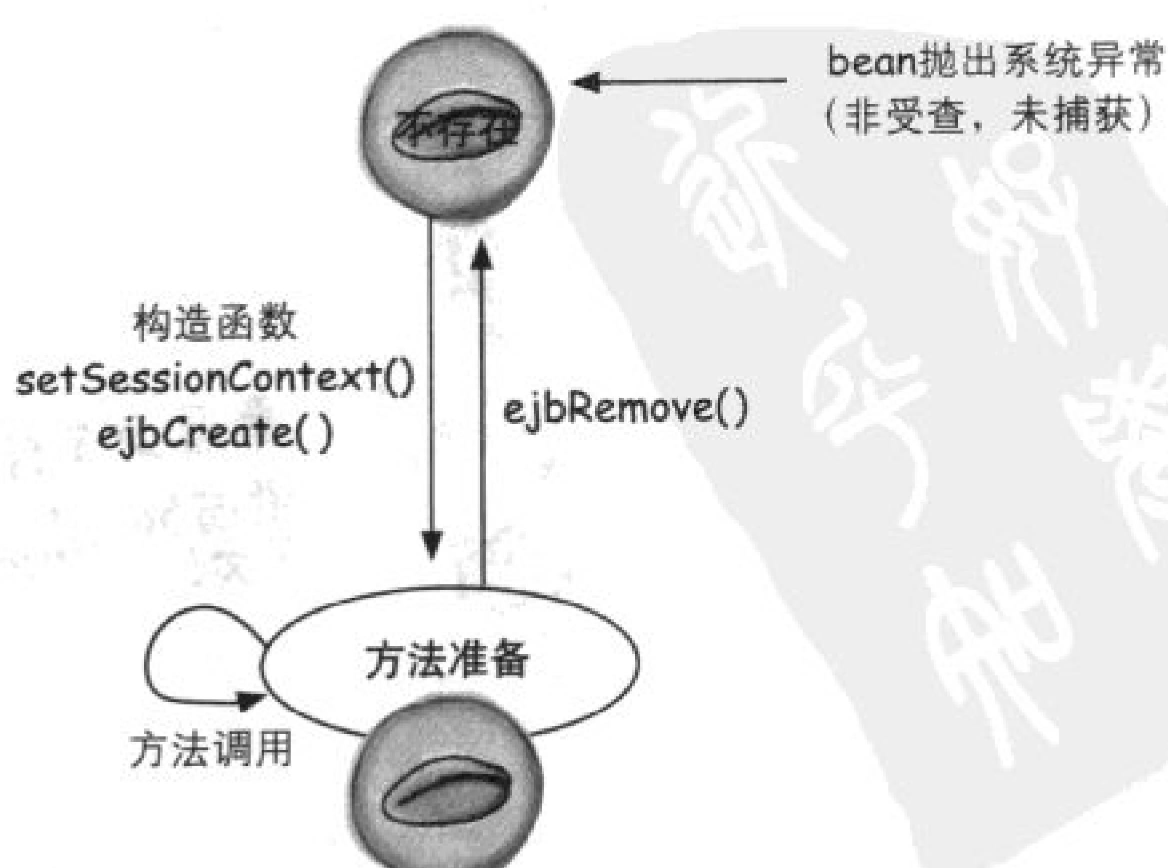
- **bean创建**（容器想建立一个bean时）
- **bean使用**（客户调用一个业务方法时）
- **bean删除**（容器认为池里有过多bean，或者bean抛出一个系统异常时）

# 比较有状态会话bean和无状态会话bean的生命周期

## 有状态

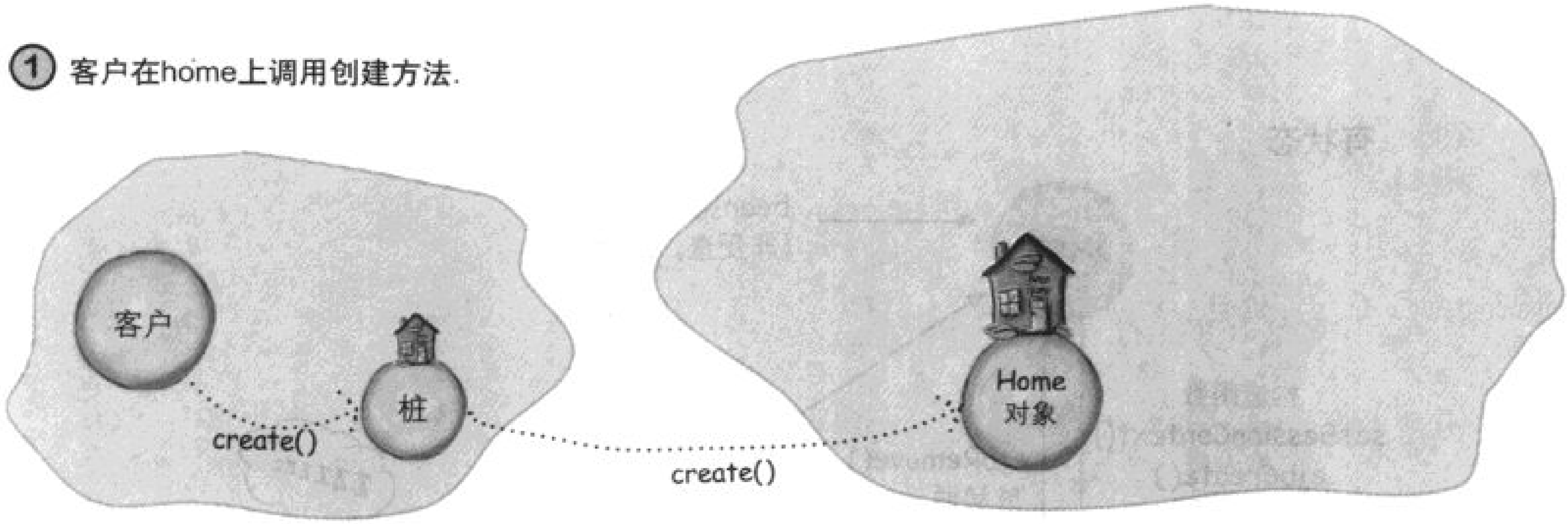



## 无状态

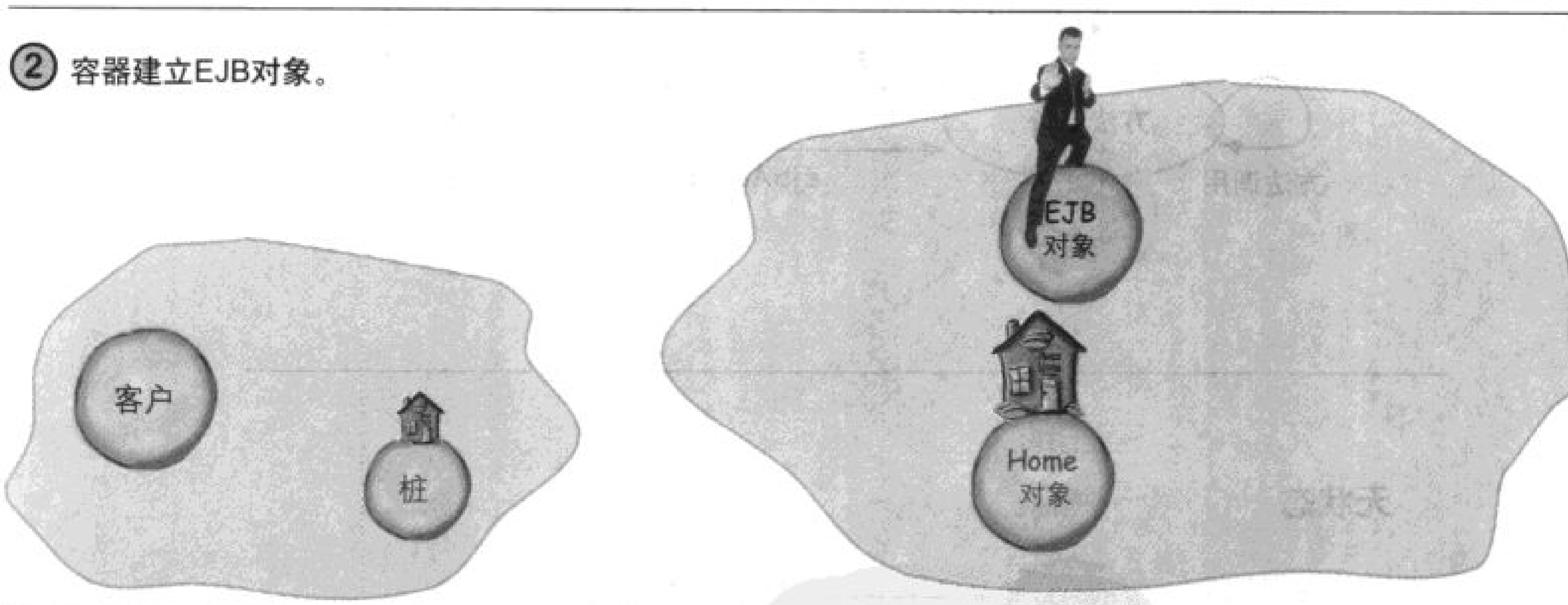


## 客户在无状态会话bean home上调用创建方法

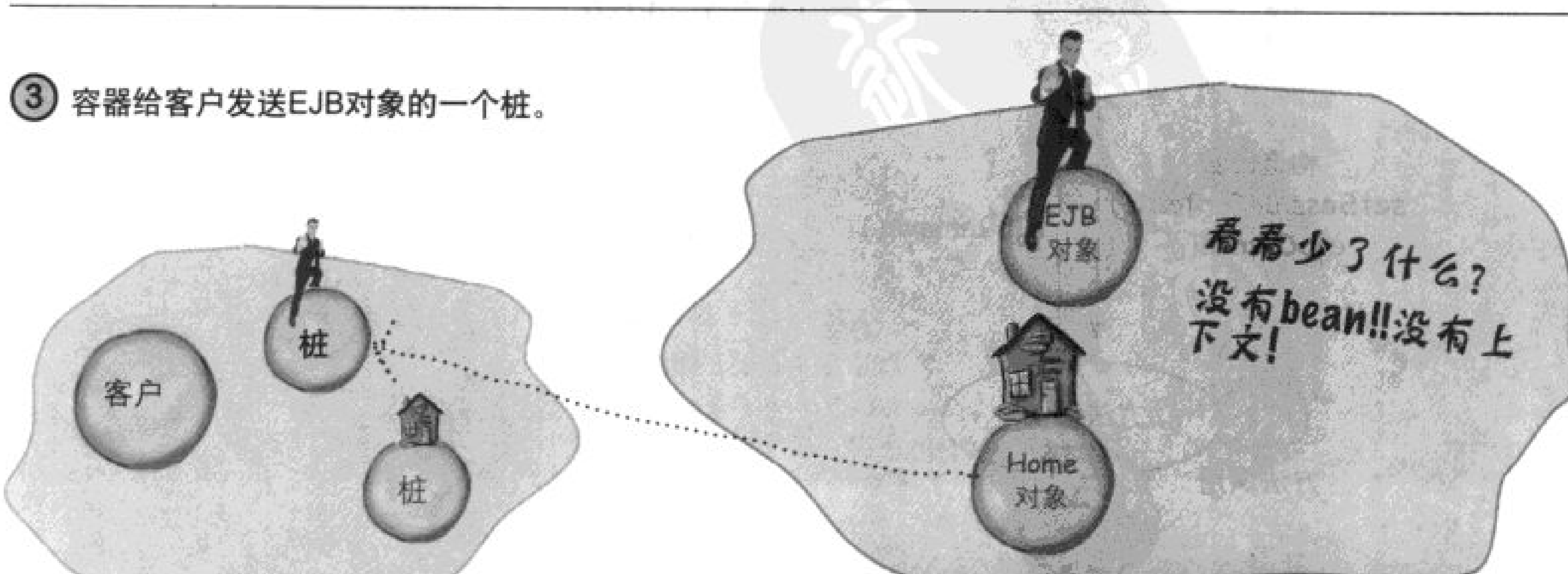
① 客户在home上调用创建方法.




② 容器建立EJB对象。

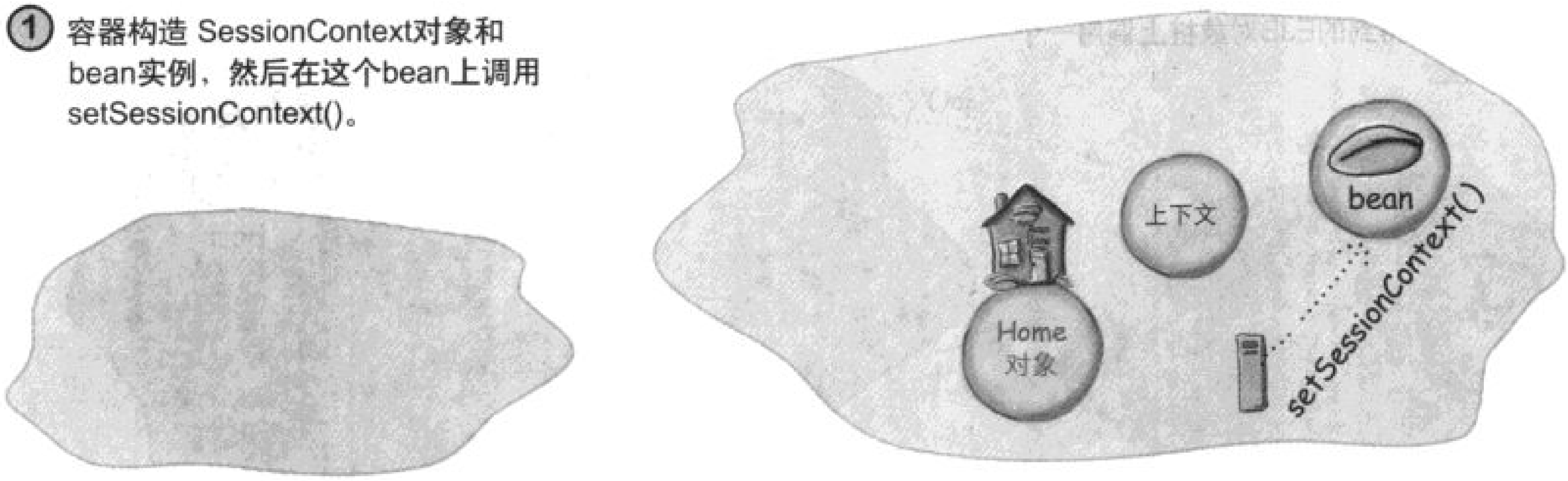



③ 容器给客户发送EJB对象的一个桩。

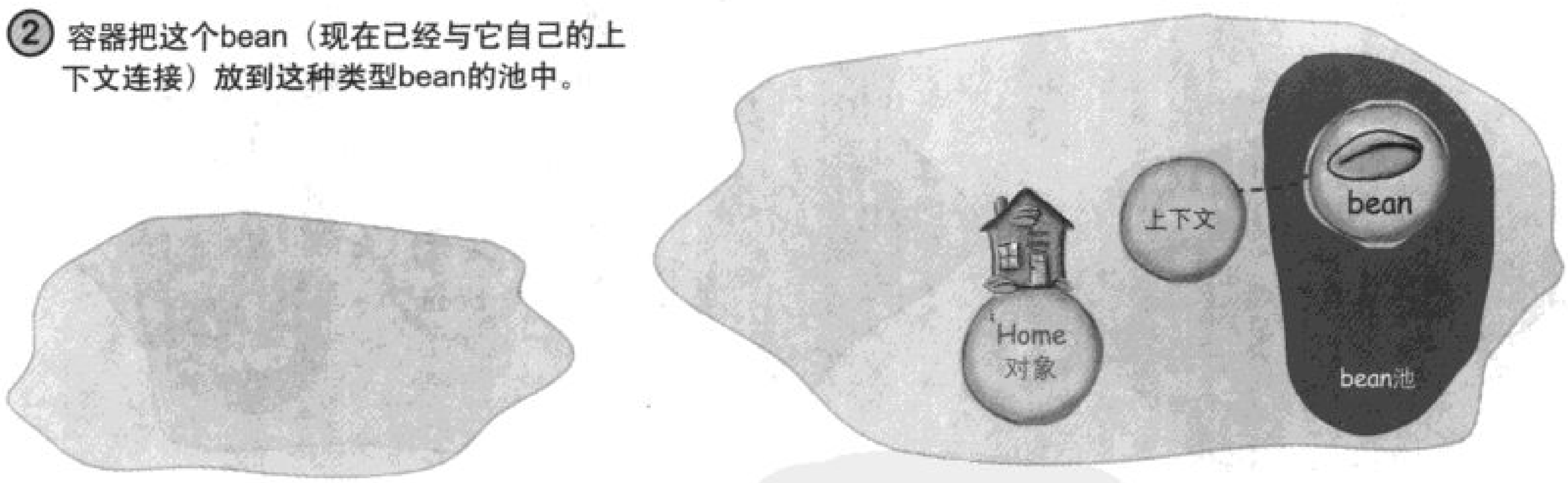


## 会话bean创建与客户无关

① 容器构造 SessionContext对象和bean实例，然后在这个bean上调用setSessionContext()。

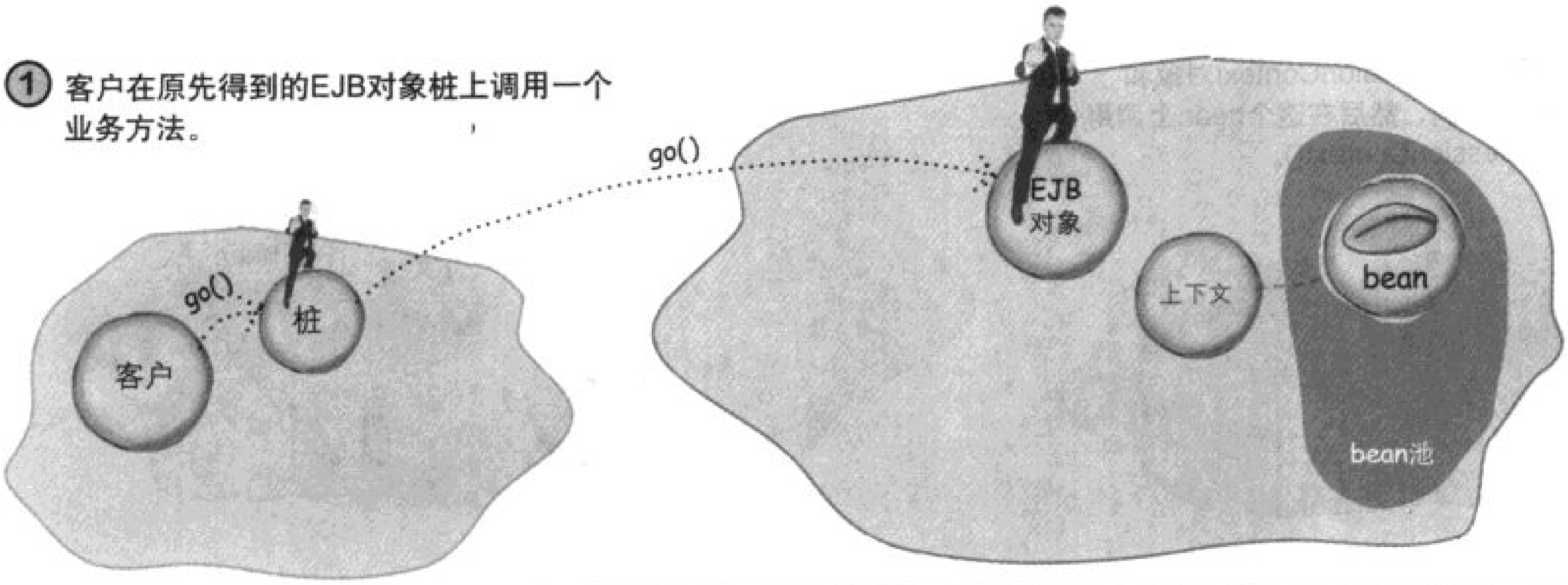



② 容器把这个bean（现在已经与它自己的上下文连接）放到这种类型bean的池中。

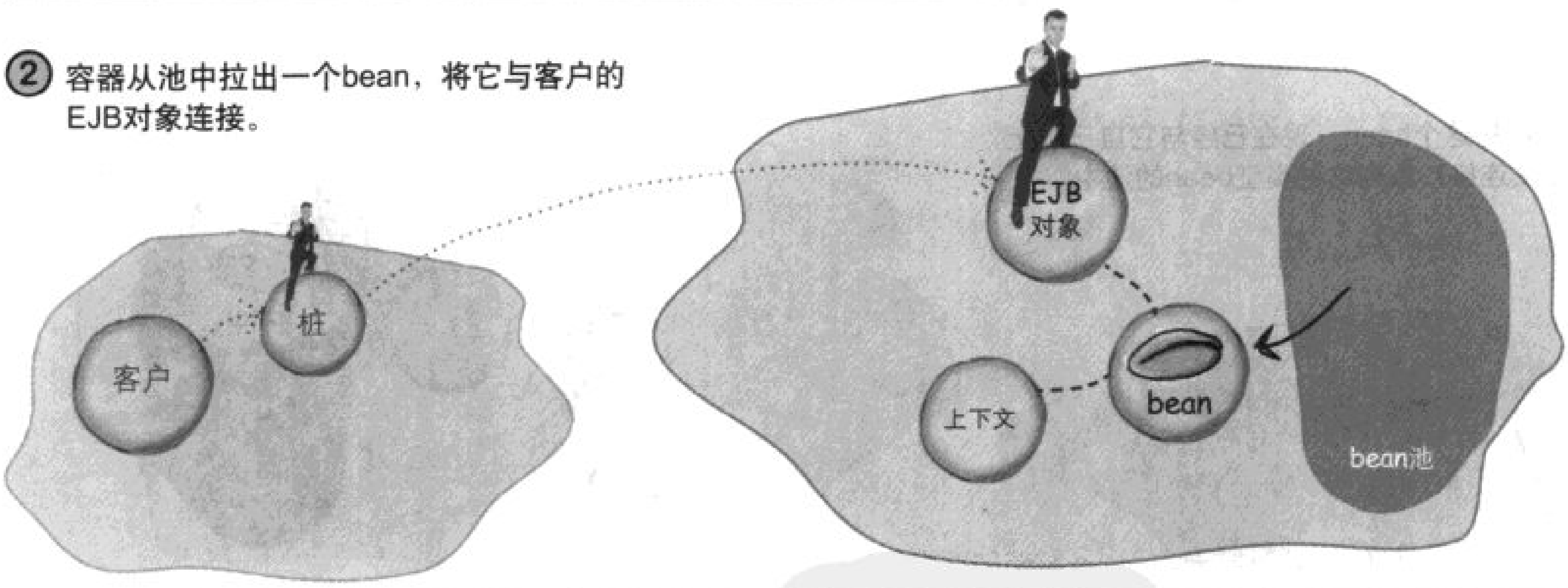


## 业务方法调用

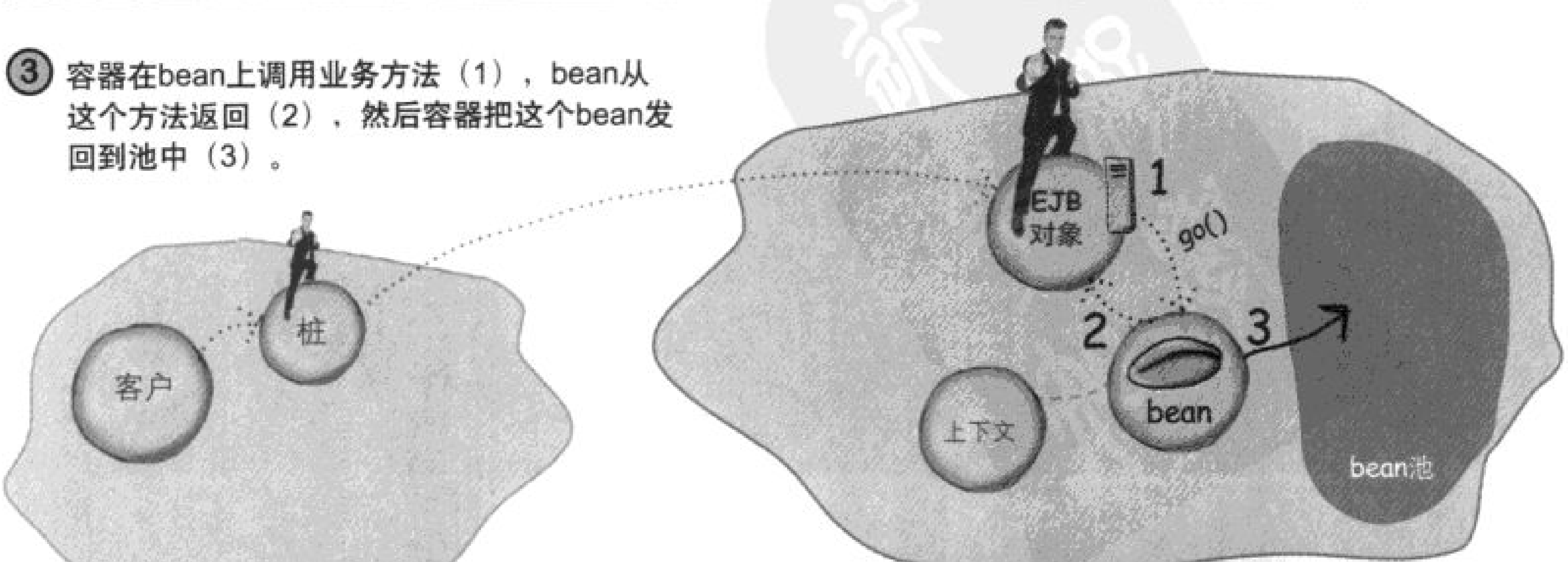
① 客户在原先得到的EJB对象桩上调用一个业务方法。




② 容器从池中拉出一个bean，将它与客户的EJB对象连接。

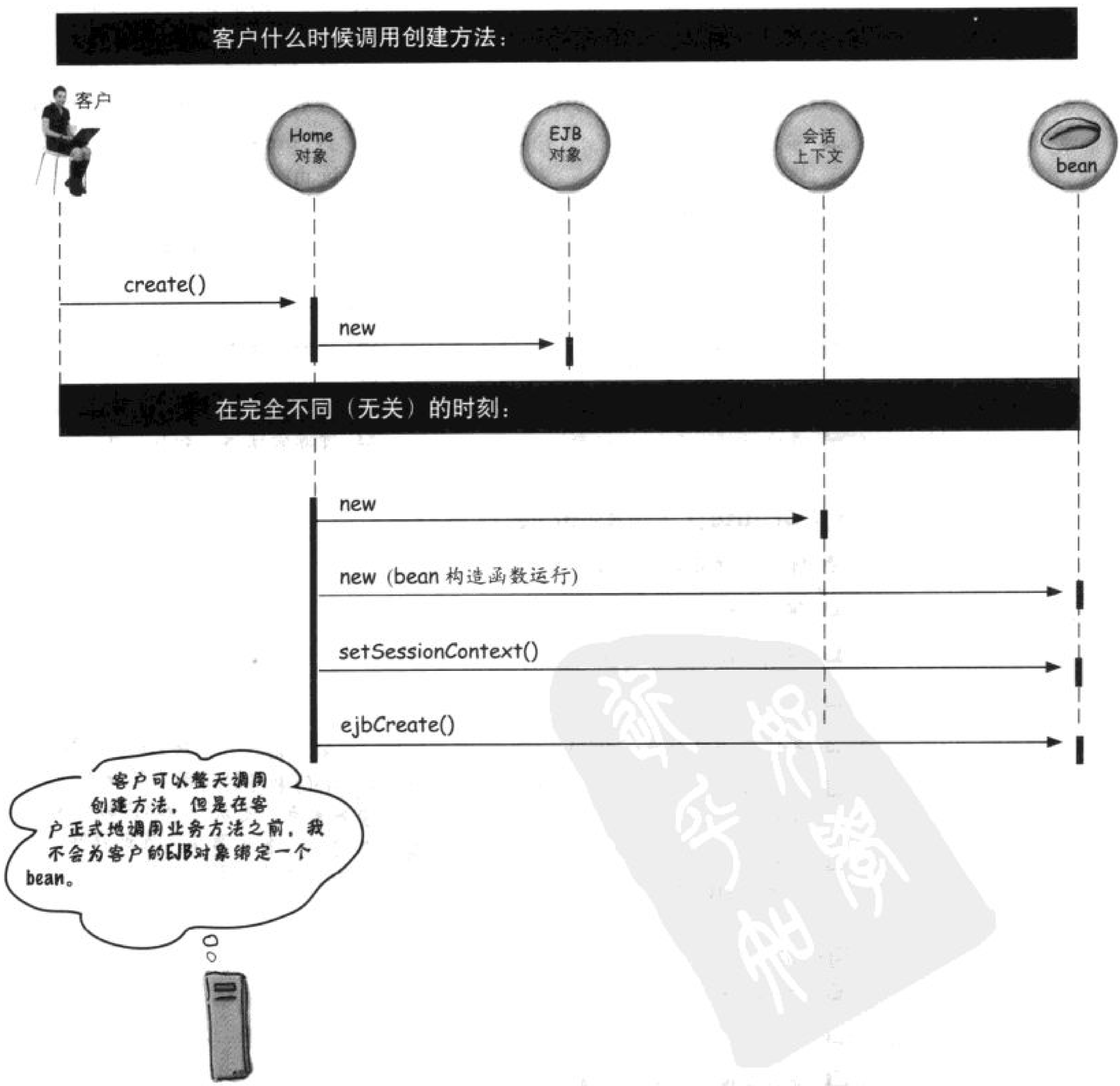



③ 容器在bean上调用业务方法（1），bean从这个方法返回（2），然后容器把这个bean发回到池中（3）。



## bean创建的对象交互图(OID)

### 无状态会话bean





# 在无状态bean的方法中能做的bean工作

## 构造函数

- 无

## setSessionContext()

使用你的SessionContext可以做：

- [x] 得到home的一个引用
- [ ] 得到EJB对象的一个引用
- [ ] 得到有关客户的安全信息
- [ ] 要求一个事务回滚（CMT bean）
- [ ] 确定事务是否已经置为回滚（CMT bean）
- [ ] 得到一个事务引用，并在上面调用方法（BMT bean）

可以访问：

- [x] 你的特殊JNDI环境
- [ ] 另一个bean的方法
- [ ] 资源管理器（如数据库）

## 业务方法

使用你的SessionContext可以做：

- [x] 得到home的一个引用
- [x] 得到EJB对象的一个引用
- [x] 得到有关客户的安全信息
- [x] 要求一个事务回滚（CMT bean）
- [x] 确定事务是否已经置为回滚（CMT bean）
- [x] 得到一个事务引用，并在上面调用方法（BMT bean）

可以访问：

- [x] 你的特殊JNDI环境
- [x] 另一个bean的方法
- [x] 资源管理器（如数据库）

## ejbCreate(), ejbRemove()

使用你的SessionContext可以做：

- [x] 得到home的一个引用
- [x] 得到EJB对象的一个引用
- [ ] 得到有关客户的安全信息
- [ ] 要求一个事务回滚（CMT bean）
- [ ] 确定事务是否已经置为回滚（CMT bean）
- [ ] 得到一个事务引用，并在上面调用方法（BMT bean）

可以访问：

- [x] 你的特殊JNDI环境
- [ ] 另一个bean的方法
- [ ] 资源管理器（如数据库）

不同于有状态bean，无状态bean不能在ejbCreate()中得到调用者的安全信息，因为客户与无状态bean的创建无关。要记住，是容器来决定创建bean，而它未与任何客户绑定。


bean在池中出出进进的时候为什么不调用ejbPassivate()和ejbActivate()?
1
没这个必要！bean在池里的时候一直是醒着的。出于性能考虑，我让bean都作为堆中的活动对象，而不是串行化对象，要不然，每次客户调用一个方法，而我需要一个bean来提供服务时，都不得不重新激活它。
2
那么，如果池中的所有bean都占用堆上的空间，你怎么得到可扩展性呢？
3
记住，对于无状态bean，不需要每个客户都有一个bean，只要正在调用一个业务方法的客户有一个bean就行了。换句话说，我只需要有足够的无状态bean来为确实在执行的方法提供服务，而不是每个有EJB对象引用的客户都要一个bean！与方法调用中实际花的时间相比，客户方法调用之间的间隔时间要多得多。

# 编写一个会话bean：这是bean提供者的任务




## 在bean类中要放上3类方法：



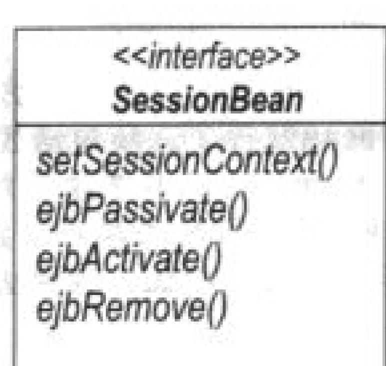

① **HOME方面：ejbCreate()方法**

在bean类中写ejbCreate()方法，分别与home接口中的各个create()方法匹配。




② **组件方面：业务方法**

在bean类中编写业务方法，分别与bean组件接口中的各个方法匹配。




③ **会话bean工作：SessionBean接口的容器回调**

实现SessionBean接口中的4个方法，你要bean必须以正式的Java方式实现这个接口（也就是说，在bean类或者它的某个超类中使用implements SessionBean声明）。

**Sharpen your pencil**

在放到bean中的三种方法中，选出哪些方法是编译器要检查的。

**编译器要检查的方法**

- ❑ 与Home接口匹配的方法
- ❑ 与组件接口匹配的方法
- ❑ SessionBean接口中的方法

**Sharpen your pencil**

给定以下接口，在这一页的下面写出bean类代码（方法可以为空）。特别要注意home创建方法……在bean中需要什么与之“匹配”？要有相同的返回类型吗？提示在这一页最底下。

```
import javax.ejb.*;
import java.rmi.RemoteException;

public interface KennelHome extends EJBHome {
   public Kennel create(String custID) throws CreateException, RemoteException;
}
```

```
import javax.ejb.*;
import java.rmi.RemoteException;

public interface Kennel extends EJBObject {
   public KennelLease placePet(Pet p) throws RemoteException;
   public void renewLease(KLease lease) throws RemoteException, ExpiredException;
   public Pet getPet(KLease lease) throws RemoteException, DeadPetException;
}
```

**在这里写出bean类：**

提示：bean中的home方法与home接口中的相应方法名字上稍有差别。容器不需要在创建bean时返回任何东西。Java中不是有一条规则说“要实现接口，必须在实现类中声明同样的异常，也就是在接口中的方法上声明的异常”还记得吗？



## HOME方法规则：home接口

❶ 本地home接口必须返回本地组件接口，而远程home接口必须返回远程组件接口。

```
public interface AdviceLocalHome extends EJBLocalHome {
  public AdviceLocal create() throws CreateException;
}

public interface AdviceHome extends EJBHome {
  public Advice create() throws CreateException,
                                    RemoteException;
}
```

❷ home接口中的每个创建方法都必须声明一个CreateException异常，而不论这个接口是本地接口还是远程接口。你还可以声明你自己的应用（受查）异常。

```
public Advice create() throws CreateException,
                                    NoAdviceException;
```

❸ 本地home接口必须扩展EJBLocalHome，而且不能声明RemoteException。

```
public interface AdviceLocalHome extends EJBLocalHome {
  public AdviceLocal create() throws CreateException;
}
```

❹ 远程home接口必须扩展EJBHome，而且其中每个方法都必须声明RemoteException异常。

```
public interface AdviceHome extends EJBHome {
  public Advice create() throws CreateException,
                                RemoteException;
}
```

❺ 无状态会话bean只能有一个create()方法，而且这个创建方法不能有参数！

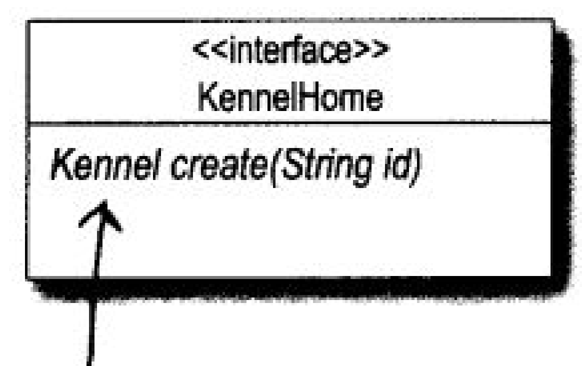
```
public Advice create()throws CreateException;
```

❻ 有状态bean必须有一个或多个create()方法，而且不一定非得有一个无参数create()。create()方法必须以串“create”开头，而且可以重载。

```
Foo createBigFoo()
Foo create()
```

❼ 远程home接口方法的参数和返回类型都必须是合法的RMI-IIOP类型（Serializable、基本类型、Remote对象或这些类型组成的数组或集合）。

**home创建方法的返回类型必定是组件接口类型。**




这必定是组件接口类型。远程home必须返回远程组件接口引用，本地home必须返回本地组件接口引用。



## HOME方法规则：bean类

1. home中的每个创建方法在bean类中都必须有一个匹配的ejbCreate方法。bean中的ejbCreate方法必须有void返回类型。

```
public void ejbCreate() { }
```

2. ejbCreate方法必须是公共（public），而且不能标记为 final 或static。

3. home接口中不一定要声明异常，除非你从你自己的方法确实抛出了这些异常，不过，声明CreateException通常是一种很好的做法。在bean类中绝对不要声明RemoteException。

```
public void ejbCreate() { } // 没有声明异常，
// 因为这里没有抛出任何异常
```

4. 如果home接口中没有为匹配的创建方法声明应用异常（即受查异常），在bean类中就不要声明这些异常。

```
public void ejbCreate() throws FireException { }
// 除非在home接口创建方法上声明了FireException，
// 否则这是非法的！！
```

5. 无状态bean只能有一个ejbCreate()方法，而且它不任何参数。

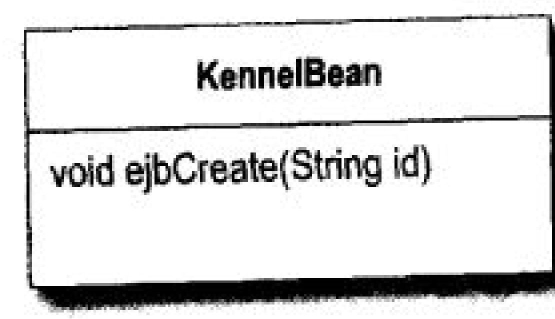
```
public void ejbCreate() { } // 必须像这样!
```

6. 有状态bean必须有一个或多个ejbCreate方法（与home接口的创建方法匹配），而且必须以串“ejbCreate”开头。

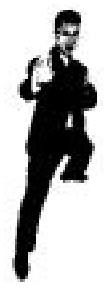
HOME:
```
public Foo createBigFoo() throws CreateException;
```

BEAN:
```
public void ejbCreateBigFoo() { }
```

home中的每个创建方法在bean类中都必须一个匹配的ejbCreate方法（返回类型为void）。




bean home方法必须与home接口方法匹配，只不过bean类方法都有一个前缀“ejb”，而且create中的字母“c”要大写，所以要以“ejbCreate”开头。



## 业务方法规则：组件接口

① 业务方法名不能以字符串“ejb”开头。

② 远程组件接口方法的参数和返回类型都必须是合法的RMI-IIOP类型。这说明，必须是Serializable、基本类型、Remote对象，或这些类型组成的数组或集合（只要集合实现类本身是Serializable）。

```
public String getAdvice()throws RemoteException;
// String 是 Serializable，所以这个方法可以
// 放在一个远程接口中。

public Socket getTheSocket();
// Socket 不是Serializable，所以这个方法
// 不能放在远程接口中（但本地接口可以）
```

**远程组件接口方法必须以RMI-IIOP类型作为参数和返回类型。**

```
String doSomething(int i)
```

这个方法在远程或本地接口中都是合法的，因为参数和返回类型都是合法的RMI-IIOP类型。

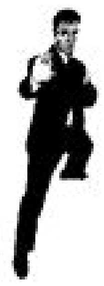
③ 本地组件接口必须扩展EJBLocalObject，而且不能声明RemoteException。

```
public interface AdviceLocal extends EJBLocalObject {
  public String getAdvice();
}
```

④ 远程组件接口必须扩展EJBObject，而且每个方法都必须声明RemoteException异常。

```
public interface Advice extends EJBObject{
  public String getAdvice() throws RemoteException;
}
```

⑤ 不能通过一个远程组件接口方法来展现一个bean的本地home或组件接口。换句话说，不能把本地接口作为远程组件接口方法的返回类型或参数类型。



## 业务方法规则：bean类

1. 业务方法必须声明为公共（public），而且不能声明为final 或static。

```
public void doBigThings() { }
```

2. 方法名不能以“ejb”开头。

3. 不能把“this”作为参数或返回值来传递。要记住，你并不想把bean的引用交出去！所有人都必须通过EJB对象这一关（在后面，我们将看到如何做到）。

```
public void doStuff(this);  // 不合法
// 从bean方法传递this不合法!
```

4. 远程组件接口方法的参数和返回类型都必须是合法的RMI-IIOP类型。这说明，必须是Serializable、基本类型、Remote对象,或这些类型组成的数组或集合（只要集合实现类本身是Serializable）。

```
public String getAdvice() { }
// String 是 Serializable，所以这个方法可以
// 放在一个远程组件接口中。

public Socket getTheSocket() { }
// Socket 不是Serializable，所以这个方法
// 不能放在远程接口中（但本地接口可以）
```

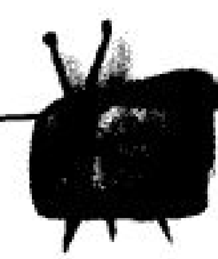
5. 不必声明组件接口中声明的异常，除非确实从你自己的方法中抛出了这些异常。在你的bean类绝不要声明RemoteException。

```
public String getAdvice() { } // 没有声明异常，
// 因为这里没有抛出任何异常。
```

6. 如果在组件接口的匹配业务方法上没有声明某个应用异常（即受查异常），就不能在bean类中声明这些这些异常。

```
public void getAdvice() throws AdviceException { }
// 除非在组件接口中的相应方法上声明了AdviceException，
// 否则这是非法的!!
```

**业务方法不能抛出未在组件接口中声明的应用（受查）异常。**



**SessionBean 扩展了 EnterpriseBean**

这里没什么特别的，不过……EnterpriseBean是Serializable，所以这意味着你的bean也自动成为Serializable，而不用明白地说明。如果给你一种场景，要求你判断一个bean是不是Serializable，答案是肯定的，尽管你并没有直接看到bean类说“implements Serializable”。



## bean 类的其他规则

① bean类必须实现javax.ejb.SessionBean，可以直接实现，也可以间接实现。

```
public class AdviceBean implements SessionBean
```

② 这个类必须是public，而且不能是final 或abstract。

③ 这个类必须有一个公共的无参数构造函数（容器必须使用这个构造函数来构造对象）。建议你就让编译器来自动插入默认构造函数，因为你并不希望在构造函数中放上代码。

```
public Advice() { }  // 合法的bean构造函数
```

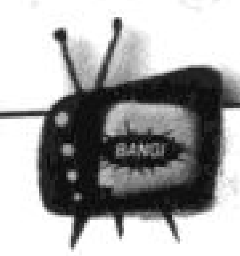
④ bean类中不能有finalize()方法！

⑤ 这个类不要实现bean的组件接口（但这在语法上是允许的）。（应该记得，我们前面讨论过为什么不想“正式”地实现组件接口。）

⑥ 这个类必须实现匹配的home和组件接口方法（也就是说，对于home中的每个创建方法都要实现一个ejbCreate，对于组件接口中的每个方法都要实现一个匹配的业务方法）。

⑦ 如果bean是有状态的，它可以（可选）实现SessionSynchronization接口（这个接口给了bean3个与事务有关的回调方法，我们将在关于事务的一章再做介绍）。

⑧ bean类可以有一个超类。换句话说，可以对bean类使用正常的Java继承。（但是要记住，还得不到任何特定于bean的继承。）



**bean类还可以有其他方法**

你的bean类必须有3种类型的方法：home方面的方法（与create匹配的ejbCreate）、组件接口的业务方法，以及4个SessionBean方法。不过你完全可以有自己的不对外展现的“辅助”方法，而且这些方法不必是public方法。

## create 和 ejbCreate之间的区别

客户需要得到组件接口

home 接口

```
public Kennel create() throws CreateException, RemoteException,PetException;
```

不要忘了名字上的区别！

只是因为接口声明了一个异常并不表示你也必须声明这个异常。如果你没有抛出这个异常，就不用声明。

bean 类

```
public void ejbCreate() throws PetException { ... }
```

但是容器并不需要从bean得到任何东西



**ejbCreate()里可以有一个return。**

你知道ejbCreate()方法的返回类型必须是void，但是别忘了基本的Java规则：如果你看到这样一条返回语句：`return;`，这并不说明你在返回一个值。不要被蒙骗，认为要返回其他东西，而不是ejbCreate()方法实际声明的返回类型（void）。

**要注意前缀“ejb”，如ejbRemove就有，而remove就没有，再比如ejbCreate和create。**

是非题：如果有状态bean处于活动状态时超时，容器会调用bean的remove()方法。

错！bean没有remove()方法，只有一个ejbRemove()方法。

注意这样一些说法“bean提供者要负责把create()方法放在bean中”。这是不对的，因为bean里根本没有create()方法，只有一个ejbCreate()。

要牢牢记住方法名以“ejb”开头的方法都在bean中！当然，要是只有这么一条规则那就简单了……比如bean中的所有方法都以“ejb”开头，而接口中所有方法都不以“ejb”开头，但当然情况并不是这样。

业务方法在组件接口和bean类中的名字都一样，而且不管在组件接口还是在bean类中，它们都不以“ejb”开头。

还需要记住谁调用什么。例如，谁调用create()方法？客户还是容器？（客户）谁调用ejbCreate()？（容器）谁调用setSessionContext()？（容器）谁调用remove()？（客户）

**一次只有一个客户可以访问一个会话bean，而不论这个bean是有状态的还是无状态的！**

会话bean不能被多个客户同时访问。如果一个bean当前正在被一个客户使用，而另一个客户又调用了这个bean的一个方法，那么第二个客户就会得到一个异常！但是……规范中有一条注释，指出容器可以把无状态bean实现为一个bean实例，这样容器就可以一次通过一个bean运行多个客户（即为多个客户服务）。

你不用知道也不必关心容器是怎么做到的：你要知道的只是，要保护会话bean不被客户并发访问，无需你自己编写代码［这你可能知道，因为你很清楚在你的bean中甚至不能用与线程相关的代码，如synchronized（同步）、wait（等待）、notify（通知）等等］。

## Sharpen your pencil

### 谁做什么?

根据下面给出的短句，按照bean提供者、容器或客户各自的职责把这些短句组织为相应的列表。

创建Home对象类

实现SessionContext类

调用setSessionContext()

调用ejbCreate()

调用ejbRemove()

实现EJBObject类

创建home接口

调用create()

实现Handle类

实现SessionBean

调用组件接口上的业务方法

实现create()方法

实现ejbCreate()方法

实现ejbActivate()方法

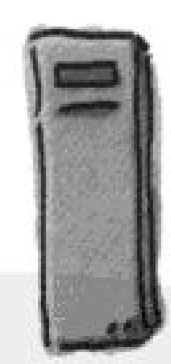
调用ejbPassivate()



bean提供者



客户



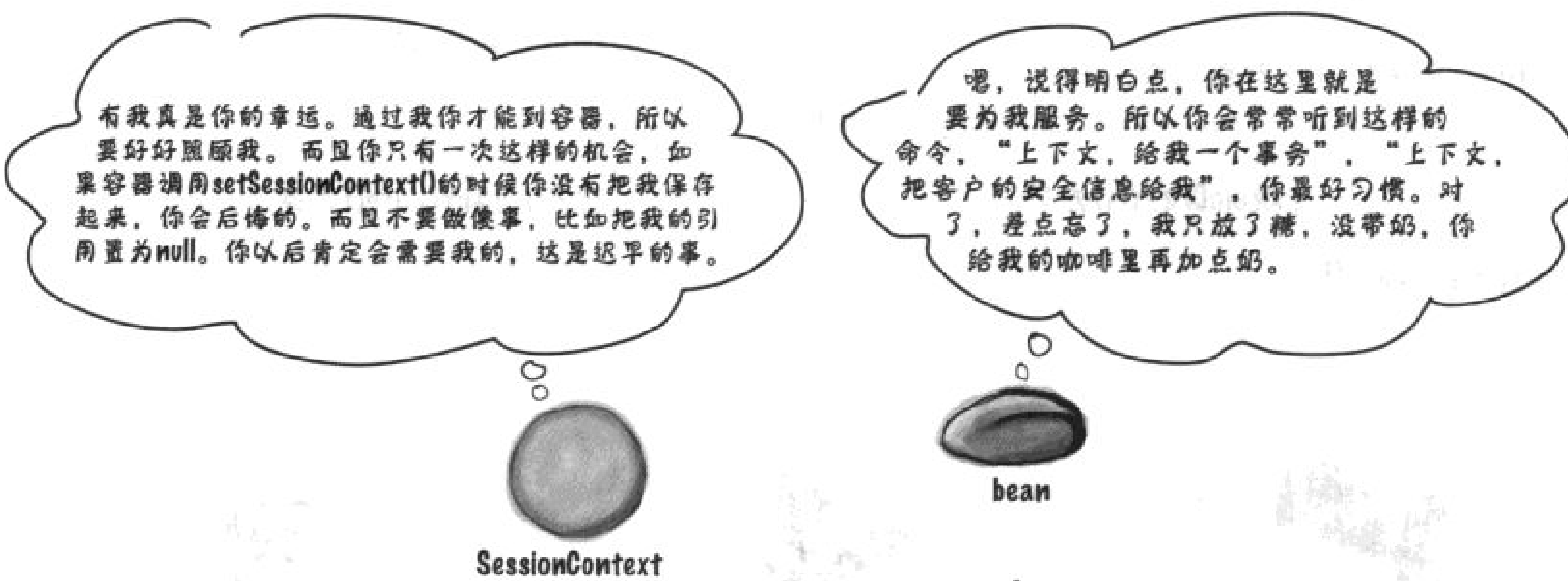
容器

# SessionContext

## 你需要它，超过了它需要你

在bean的生命之初，而且在bean的一生中也只有这么一次，容器会调用bean的setSessionContext()容器回调方法。

这个上下文（这是EJBContext的一个子类）是bean到容器的生命线，bean只能调用这个上下文的方法来得到bean自己的home和EJB对象的引用，得到有关客户的安全信息，或者做与事务相关的事情。现在先不看安全和事务信息；这些内容会在相应的章节里介绍。




**记住SessionContext接口！**

你要会看方法，给你一组方法，你要知道哪些方法属于SessionContext接口。而且要记住，与生命周期有关的许多问题都是关于SessionContext接口方法，也就是要知道在哪些方法中可以调用上下文的某些方法。例如，你要知道不能在无状态bean的ejbCreate()方法中从这个上下文得到与客户相关的信息，这是因为，没有任何客户与无状态bean的ejbCreate()方法关联！但是，可以在一个有状态bean的ejbCreate()方法中从这个上下文得到客户信息，因为这个有状态bean正是一个客户要求建立的。

## SessionContext能做的事情
## (9种方法，分为4类)

**home引用**

- 得到home的一个引用

  **getEJBHome()**
  **getEJBLocalHome()**

---

**EJB 对象引用**

- 得到EJB对象的一个引用

  **getEJBObject()**
  **getEJBLocalObject()**

---

**客户安全性**

- 得到有关客户的安全信息

  **getCallerPrincipal()**
  **isCallerInRole(String s)**

---

**事务**

- 要求一个事务回滚（仅CMT）

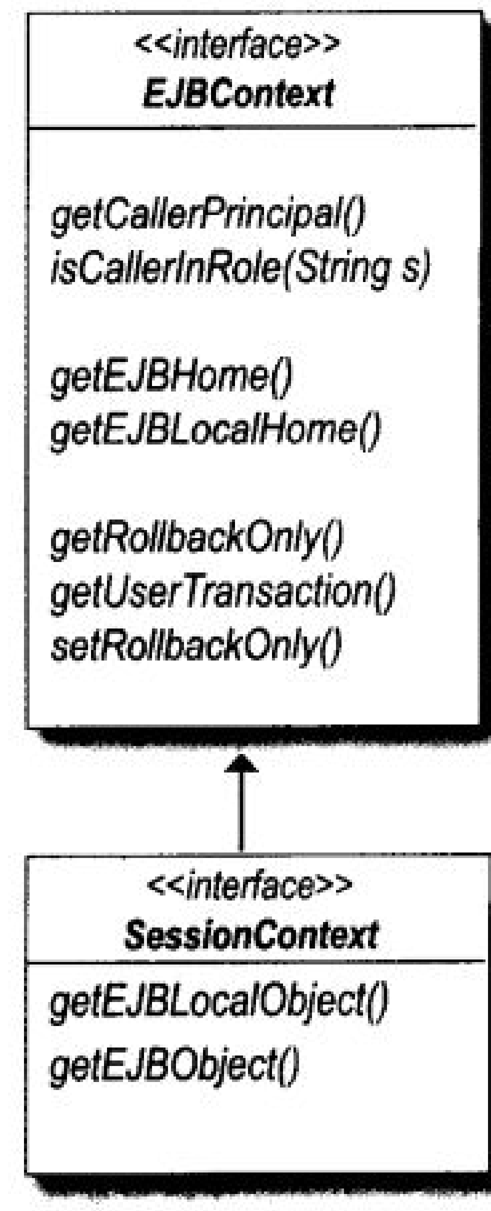
  **setRollbackOnly(boolean b)**

- 确定事务是否已经置为回滚（仅CMT）

  **getRollbackOnly()**

- 得到一个事务引用，并开始你自己的事务（仅BMT）

  **getUserTransaction()**




**Relax** **你不必知道这3个废弃的方法。**

除了这一页上列的方法，EJBContext接口还有另外3个方法，这些方法都已经废弃了。这些废弃的方法在考试中不会考，所以如果在API中看到它们，不要害怕。之所以让你了解这一点，只是因为万一你要处理比较老的EJB代码。这些方法包括：

isCallerInRole(Identity id)

getCallerIdentity()

getEnvironment()



**无状态bean**：真不知道你怎么能受得了，就这样当个有状态bean。你的一辈子就是要服侍一个客户。

**有状态bean**：我不太喜欢你这么说，什么“服侍一个客户”。听起来好像我是一个奴隶，一个仆人一样。

**无状态bean**：那你觉得这样说不对吗？为什么不对？

**有状态bean**：这是一种私人的、亲密的关系。这种关系你是不会明白的。

**无状态bean**：这到底是什么意思？

**有状态bean**：这是因为你们的关系太……短暂了。你甚至不关心你是在为谁服务。照你们的话说，哪个客户都行，是不是？

**无状态bean**：这没什么不对呀。我是一个独立的人，我喜欢和别的bean待在池子里，这样总能碰上新的客户。我们的生活一点也不枯燥，而且同一个客户很少会遇见两次。

**有状态bean**：就是这样！你甚至不能实实在在地维护一个有意义、持久的关系。

**无状态bean**：什么？

**有状态bean**：这样很快就会忘记。在方法调用之间你没有办法保留任何客户信息，你为什么不关心到底是哪个客户调用，原因就在这里。就算是同一个客户反反复复地调用你的方法，你也永远记不住。

**无状态bean**：不错，但这也不算什么问题呀。要知道，用我来完成的这些服务并不需要我在方法调用之间记住客户的任何信息。

**有状态bean**：所以嘛，不能拿你去做重要的或者严肃的事情。

**无状态bean**：先等等，不是这样的！实际上，如果说到性能，我比你更可扩展。而且你也不要争，不要给我说什么钝化。这些我早就听够了。不管怎么样，再怎么钝化你也不如我那么可扩展。

**有状态bean**：好吧，也许我确实没有你可扩展……但是还有更重要的事情要考虑。

**无状态bean**：比方说？

**有状态bean**：嗯，我使用起来很容易。很容易构建，而且构建也很快。这对开发人员来说可能更重要。而且在某些情况下，也就是确实需要客户会话状态时，我还能表现得更好。有时这确实很烦人……对每个调用都必须把状态保存到一个数据库中（然后收到下一个调用的时候再在数据库里查找），是不是？尽管如此，但我很清楚有时候必须这么做。或者，当然你也可以一直向客户发回会话状态，这样客户每次调用时就能再回来找你，但是——

**无状态bean**：这样做有什么不好吗？

**有状态bean**：你不知道带宽（bandwidth）吗？好好学学吧，跟我念一遍“d-ai-dai-k-u-an-kuan”。如果这样来来回回地传递，该多浪费带宽呀，你真地想这样吗？

**无状态bean**：好吧，这我知道了。但是，给我说说看，和钝化/激活相比，这样做的话，对性能影响更大吗？而且……先等等，你听到什么了吗？好像是你的客户在叫你呢。听上去好像是……先别说话……好像是……哦，我想他是在调用remove()。很高兴和你谈话。等我去bean池的时候我会想你的。我会躺在气垫上晒着日光浴，喝着加冰酒，啊哈，我真地很高兴自己是个无状态bean。



**哪些是允许的？**

对应最左列上的方法，如果在这些方法中可以对SessionContext调用方法，请在相应的SessionContext方法上打勾。举例来说，在ejbCreate()中，无论是有状态会话bean还是无状态会话bean，你都能得到这个bean home的一个引用，所以可以在有状态会话bean和无状态会话bean这两列最上面的方框（即对应"得到home的一个引用"的方框）上打勾。所有答案都可以在这一章中找到，不过返回去找答案之前，先尽量猜猜看。你也可以把JNDI、资源和bean访问也考虑在内，自己来建立这样一个表。

| bean方法 | 有状态 | 无状态 |
|---|---|---|
| ejbCreate<br>ejbRemove | 使用你的SessionContext可以做：<br>❑ 得到home的一个引用<br>❑ 得到EJB对象的一个引用<br>❑ 得到有关客户的安全信息<br>❑ 要求一个事务回滚（CMT bean）<br>❑ 确定事务是否已经置为回滚（CMT bean）<br>❑ 得到一个事务引用，并在上面调用方法（BMT bean） | 使用你的SessionContext可以做：<br>❑ 得到home的一个引用<br>❑ 得到EJB对象的一个引用<br>❑ 得到有关客户的安全信息<br>❑ 要求一个事务回滚（CMT bean）<br>❑ 确定事务是否已经置为回滚（CMT bean）<br>❑ 得到一个事务引用，并在上面调用方法（BMT bean） |
| ejbPassivate<br>ejbActivate | 使用你的SessionContext可以做：<br>❑ 得到home的一个引用<br>❑ 得到EJB对象的一个引用<br>❑ 得到有关客户的安全信息<br>❑ 要求一个事务回滚（CMT bean）<br>❑ 确定事务是否已经置为回滚（CMT bean）<br>❑ 得到一个事务引用，并在上面调用方法（BMT bean） | 不适用：<br>无状态bean不能钝化。 |
| 业务方法 | 使用你的SessionContext可以做：<br>❑ 得到home的一个引用<br>❑ 得到EJB对象的一个引用<br>❑ 得到有关客户的安全信息<br>❑ 要求一个事务回滚（CMT bean）<br>❑ 确定事务是否已经置为回滚（CMT bean）<br>❑ 得到一个事务引用，并在上面调用方法（BMT bean） | 使用你的SessionContext可以做：<br>❑ 得到home的一个引用<br>❑ 得到EJB对象的一个引用<br>❑ 得到有关客户的安全信息<br>❑ 要求一个事务回滚（CMT bean）<br>❑ 确定事务是否已经置为回滚（CMT bean）<br>❑ 得到一个事务引用，并在上面调用方法（BMT bean） |

# COFFEE CRAM 模拟测验

**1** 关于会话bean，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 有状态会话bean通常用于表示数据库中的一行
- ❑ B. 客户在调用任何业务方法之前必须先调用ejbActivate方法
- ❑ C. 有状态会话bean的字段可以包含客户会话状态
- ❑ D. 会话bean通常用作异步消息消费者

**2** 在一个会话bean的生命周期中，下面哪几组步骤的顺序是对的？（选出所有正确的答案。）

- ❑ A. `ejbCreate(), newInstance(), setSessionContext()`
- ❑ B. `newInstance(), setSessionContext(), ejbCreate()`
- ❑ C. `ejbCreate(), setSessionContext(), newInstance()`
- ❑ D. `newInstance(), ejbCreate(), setSessionContext()`
- ❑ E. `setSessionContext(), newInstance(), ejbCreate()`

**3** 哪些类型的会话bean实例字段可以成功地钝化和重新激活？（选出所有正确的答案。）

- ❑ A. 远程home接口的引用
- ❑ B. UserTransaction接口的引用
- ❑ C. 一个数据库游标的引用
- ❑ D. SessionContext对象的引用
- ❑ E. socket的引用

**4** 下面哪些是合法的会话bean远程组件接口？（选出所有正确的答案。）

❑ A.
```
public interface MyBean extends javax.ejb.EJBObject {
    void myMethod();
}
```

❑ B.
```
public interface MyBean extends javax.ejb.EJBObject
throws RemoteException {
    void myMethod();
}
```

❑ C.
```
public interface MyBean extends
javax.ejb.EJBHomeObject {
    void myMethod() throws RemoteException;
}
```

❑ D.
```
public interface MyBean extends javax.ejb.EJBObject {
    void myMethod() throws RemoteException;
}
```

**5** 如果一个客户对已经被容器删除的会话对象做了一个调用，可能抛出哪些异常？（选出所有正确的答案。）

❑ A. `java.rmi.RemoteException`

❑ B. `javax.ejb.RemoveException`

❑ C. `java.rmi.NoSuchObjectException`

❑ D. `javax.ejb.NoSuchEntityException`

❑ E. `javax.ejb.ObjectNotFoundException`

**6** 给定：

```
15.MyBean create(String name) throws CreateException,
RemoteException;
```

哪个会话bean接口可以包含这个方法？

❑ A. 仅有状态会话bean

❑ B. 仅无状态会话bean

❑ C. 有状态会话bean和无状态会话bean都可以

❑ D. 有状态会话bean和无状态会话bean都不可以

**7** 关于bean的生命周期，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 如果一个业务方法抛出一个系统异常，这个bean会被钝化
- ❑ B. 钝化的bean在被删除之前必须先激活
- ❑ C. 无状态会话bean的删除必须由客户发起，而不是容器
- ❑ D. 无状态会话bean不能实现SessionSynchronization接口

---

**8** bean提供者必须做哪些工作才能确保会话bean成功钝化？（选出所有正确的答案。）

- ❑ A. 提供者必须调用 **`ejbPassivate()`**
- ❑ B. 如果bean是有状态的，提供者必须为**`ejbPassivate()`**方法增加业务逻辑
- ❑ C. 提供者在**`ejbPassivate()`**完成之前不必关闭所有数据库连接
- ❑ D. 提供者必须有一个假设：在标记为临时的实例字段中存储的任何状态都会丢失
- ❑ E. 提供者必须有一个假设：如果SessionContext对象不是可串行化的，那么经过钝化之后，SessionContext对象的任何引用都不能幸存

---

**9** 给定一个带容器托管事务划分的无状态会话bean，从哪些方法可以访问另一个bean？（选出所有正确的答案。）

- ❑ A. **`ejbCreate()`**
- ❑ B. **`ejbRemove()`**
- ❑ C. 业务方法
- ❑ D. **`setSessionContext()`**

**10** （注意：实际考试中有一些"拖放"类的问题，就是把某一项拖到另外一项上，这道题就是模拟这种题型……）

把左边的方法与右边的接口匹配（在哪些接口里能找到这些方法）。如果一个接口中声明了一个方法，或者一个接口继承了一个方法，就可以把该方法与这个接口匹配。注意：答案中可能有一些多对一和一对多的关系。

| | |
|---|---|
| A. afterCompletion | 1. SessionSynchronization |
| B. getUserTransaction | 2. SessionContext |
| C. afterBegin | 3. SessionBean |
| D. isCallerInRole | 4. UserTransaction |
| E. getRollbackOnly | |
| F. setSessionContext | |
| G. setRollbackOnly | |

**11** 关于会话bean的生命周期，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 客户在一个无状态bean的home接口上调用**`create()`**方法时，容器总会创建一个新的会话bean实例
- ❑ B. 客户在一个无状态bean的home接口上调用**`remove()`**方法时，容器总会调用**`ejbRemove()`**
- ❑ C. 会话bean处在一个事务中时不能钝化
- ❑ D. 无状态bean的生命周期中不会调用**`setSessionContext()`**

**12** 下面哪些方法是容器回调方法？（选出所有正确的答案。）

- ❑ A. **`ejbPassivate()`**
- ❑ B. **`setRollbackOnly()`**
- ❑ C. **`setSessionContext()`**
- ❑ D. **`getRollbackOnly()`**
- ❑ E. **`getUserTransaction()`**

**13** 在哪些情况下容器无法在会话bean上调用ejbRemove()？（选出所有正确的答案。）

- ❑ A. 如果容器崩溃
- ❑ B. 如果从一个业务方法抛出一个应用异常
- ❑ C. 如果bean处在方法状态时出现超时
- ❑ D. 如果从一个事务中抛出一个应用异常

**14** 下面哪些方法允许带bean托管事务划分的有状态会话bean和无状态会话都能访问UserTransaction方法？（选出所有正确的答案。）

- ❑ A. **ejbCreate()**
- ❑ B. **ejbRemove()**
- ❑ C. 业务方法
- ❑ D. **setSessionContext()**

**15** 对于哪种类型的bean，容器可以将处在一个事务中的实例进行钝化？

- ❑ A. 仅有状态会话bean
- ❑ B. 仅无状态会话bean
- ❑ C. 有状态会话bean和无状态会话bean都可以
- ❑ D. 有状态会话bean和无状态会话bean都不可以

**16** 对于会话bean，哪些是容器的职责？（选出所有正确的答案。）

- ❑ A. 调用本地接口**create()**方法
- ❑ B. 调用**getSessionContext()**方法
- ❑ C. 调用**ejbCreate()**方法
- ❑ D. 确保**ejbRemove()**方法总会得到调用

**17** 这是一道拖放类的题目，每一项只能用一次。各个接口应当与哪个事实匹配，才能保证所有4个匹配都正确？

| | |
|---|---|
| 1. 远程组件接口 | a. 有一个**`getHomeHandle()`**方法 |
| 2. 远程home接口 | b. 扩展‘**`javax.ejb.EJBObject`**’ |
| 3. 本地组件接口 | c. 方法不能抛出‘**`java.rmi.RemoteException`**’ |
| 4. 本地home接口 | d. 可以用于获取一个EJBLocalObject |

**18** 在一个会话bean的生命周期中，哪些方法是容器调用的？（选出所有正确的答案。）

- ❑ A. **`create()`**
- ❑ B. bean 构造函数
- ❑ C. **`setRollbackOnly()`**
- ❑ D. **`getUserTransaction()`**
- ❑ E. **`ejbRemove()`**

**19** 关于会话bean类，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 可以标记为“final”
- ❑ B. 不需要无参数构造函数
- ❑ C. 它们的组件接口方法可以是“private”
- ❑ D. 它们的业务方法名必须以“ejb”开头
- ❑ E. 它们的“ejbCreate”方法不能声明为“final”

**20** 哪种类型的bean可以实现SessionSynchronization接口？

- ❑ A. 仅有状态会话bean
- ❑ B. 仅无状态会话bean
- ❑ C. 有状态会话bean和无状态会话bean都可以
- ❑ D. 有状态会话bean和无状态会话bean都不可以

**21** 关于无状态会话bean实例，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 任何实例可以用于任何客户
- ❑ B. 会话状态必须在方法调用之间保留
- ❑ C. 会话状态必须跨事务保留
- ❑ D. 可以钝化
- ❑ E. 不支持事务

COFFEE CRAM

# 模拟测验答案

(规范：71)

**1** 关于会话bean，下面哪些说法是正确的？ （出所有正确的答案。）

☐ A. 有状态会话bean通常用于表示数据库中的一行 —实体bean才这样

☐ B. 客户在调用任何业务方法之前必须先调用ejbActivate方法 —只有容器调用ejbActivate()

☐ C. 有状态会话bean的字段可以包含客户会话状态

☑ D. 会话bean通常用作异步消息消费者

(规范：77)

**2** 在一个会话bean的生命周期中，下面哪几组步骤的顺序是对的？ （选出所有正确的答案。）

☐ A. `ejbCreate(), newInstance(), setSessionContext()`

☑ B. `newInstance(), setSessionContext(), ejbCreate()`

☐ C. `ejbCreate(), setSessionContext(), newInstance()`

☐ D. `newInstance(), ejbCreate(), setSessionContext()`

☐ E. `setSessionContext(),newInstance(),ejbCreate()`

(规范：71)

**3** 哪些类型的会话bean实例字段可以成功地钝化和重新激活？ （选出所有正确的答案。）

☑ A. 远程home接口的引用

☑ B. UserTransaction接口的引用

☐ C. 一个数据库游标的引用 —要记住，钝化可能是串行化

☑ D. SessionContext对象的引用

☐ E. socket的引用 —要记住，钝化可能是串行化

(规范：97)

**4** 下面哪些是合法的会话bean远程组件接口？（选出所有正确的答案。）

❑ A.
```
public interface MyBean extends javax.ejb.EJBObject {
    void myMethod();
}
```
—需要RemoteException

❑ B.
```
public interface MyBean extends javax.ejb.EJBObject
    throws RemoteException {
        void myMethod();
}
```
—非法Java！

❑ C.
```
public interface MyBean extends
    javax.ejb.EJBHomeObject {
        void myMethod() throws RemoteException;
}
```
—不对，应该是EJBObject

☑ D.
```
public interface MyBean extends javax.ejb.EJBObject {
    void myMethod() throws RemoteException;
}
```

(规范：79)

**5** 如果一个客户对已经被容器删除的会话对象做了一个调用，可能抛出哪些异常？(选出所有正确的答案。)

☑ A. `java.rmi.RemoteException`

❑ B. `javax.ejb.RemoveException` —这只在remove()上发生

☑ C. `java.rmi.NoSuchObjectException`

❑ D. `javax.ejb.NoSuchEntityException` —客户看不到这个异常

❑ E. `javax.ejb.ObjectNotFoundException` —这是实体查找方法抛出的异常

(规范：98)

**6** 给定：

```
MyBean create(String name) throws CreateException, RemoteException;
```

哪个会话bean接口可以包含这个方法？

☑ A. 仅有状态会话bean

❑ B. 仅无状态会话bean

❑ C. 有状态会话bean和无状态会话bean都可以

❑ D. 有状态会话bean和无状态会话bean都不可以

无状态会话bean只能有无参数创建方法！

(规范：79，88)

**7** 关于bean的生命周期，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 如果一个业务方法抛出一个系统异常，这个bean会被钝化 — 不，bean会被删除
- ❑ B. 钝化的bean在被删除之前必须先激活 — 不对
- ❑ C. 无状态会话bean的删除必须由客户发起，而不是容器
- ☑ D. 无状态会话bean不能实现SessionSynchronization接口

(规范：71-72)

**8** bean提供者必须做哪些工作才能确保会话bean成功钝化？（选出所有正确的答案。）

- ❑ A. 提供者必须调用**`ejbPassivate()`**
- ❑ B. 如果bean是有状态的，提供者必须为**`ejbPassivate()`**方法增加业务逻辑
- ❑ C. 提供者在**`ejbPassivate()`**完成之前不必关闭所有数据库连接
- ☑ D. 提供者必须有一个假设：在标记为临时的实例字段中存储的任何状态都会丢失
- ❑ E. 提供者必须有一个假设：如果SessionContext对象不是可串行化的，那么经过钝化之后，SessionContext对象的任何引用都不能幸存

容器必须钝化你的上下文，不管怎么钝化！

(规范：90)

**9** 给定一个带容器托管事务划分的无状态会话bean，从哪些方法可以访问另一个bean？（选出所有正确的答案。）

- ❑ A. **`ejbCreate()`**
- ❑ B. **`ejbRemove()`**
- ☑ C. 业务方法 — 对无状态BMT bean也成立
- ❑ D. **`setSessionContext()`**

**10** (注意：实际考试中有一些“拖放”类的问题，就是把某一项拖到另外一项上，这道题就是模拟这种题型……) (API文档)

把左边的方法与右边的接口匹配（在哪些接口里能找到这些方法）。如果一个接口中声明了一个方法，或者一个接口继承了一个方法，就可以把该方法与这个接口匹配。注意：答案中可能有一些多对一和一对多的关系。

| | | |
|---|---|---|
| A. afterCompletion | 1 | 1. SessionSynchronization |
| B. getUserTransaction | 2 | 2. SessionContext |
| C. afterBegin | 1 | 3. SessionBean |
| D. isCallerInRole | 2 | 4. UserTransaction |
| E. getRollbackOnly | 2 | |
| F. setSessionContext | 3 | |
| G. setRollbackOnly | 2, 4 | |

**11** 关于会话bean的生命周期，下面哪些说法是正确的？（选出所有正确的答案。） (规范：70)

- ❑ A. 客户在一个无状态bean的home接口上调用**create()**方法时，容器总会创建一个新的会话bean实例 — 只要容器想，什么时候都可以建立无状态会话bean
- ❑ B. 客户在一个无状态bean的home接口上调用**remove()**方法时，容器总会调用**ejbRemove()** — 容器只在想缩减bean池的时候才会删除一个无状态会话bean
- ☑ C. 会话bean处在一个事务中时不能钝化
- ❑ D. 无状态bean的生命周期中不会调用**setSessionContext()**

**12** 下面哪些方法是容器回调方法？（选出所有正确的答案。） (规范：73)

- ☑ A. **ejbPassivate()**
- ❑ B. **setRollbackOnly()**
- ☑ C. **setSessionContext()**
- ❑ D. **getRollbackOnly()**
- ❑ E. **getUserTransaction()**

**13** 在哪些情况下容器无法在会话bean上调用ejbRemove()？（选出所有正确的答案。） （规范：82）

- ☑ A. 如果容器崩溃
- ☐ B. 如果从一个业务方法抛出一个应用异常
- ☐ C. 如果bean处在方法状态时出现超时
- ☐ D. 如果从一个事务中抛出一个应用异常

---

**14** 下面哪些方法允许带bean托管事务划分的有状态会话bean和无状态会话都能访问UserTransaction方法？（选出所有正确的答案。） （规范：80，90）

- ☐ A. **`ejbCreate()`** — 有状态会话bean才可以
- ☐ B. **`ejbRemove()`**
- ☑ C. 业务方法
- ☐ D. **`setSessionContext()`**

---

**15** 对于哪种类型的bean，容器可以将处在一个事务中的实例进行钝化？ （规范：70）

- ☐ A. 仅有状态会话bean
- ☐ B. 仅无状态会话bean
- ☐ C. 有状态会话bean和无状态会话bean都可以
- ☑ D. 有状态会话bean和无状态会话bean都不可以

---

**16** 对于会话bean，哪些是容器的职责？（选出所有正确的答案。）

- ☐ A. 调用本地接口**`create()`**方法 —是客户调用
- ☐ B. 调用**`getSessionContext()`**方法 —容器调用setSessionContext
- ☑ C. 调用**`ejbCreate()`**方法
- ☐ D. 确保**`ejbRemove()`**方法总会得到调用 — 如果钝化bean超时，或者bean抛出一个运行时异常，就不会调用这个方法

**17** 这是一道拖放类的题目，每一项只能用一次。各个接口应当与哪个事实匹配，才能保证所有4个匹配都正确？

| | | |
|---|---|---|
| 1. 远程组件接口 | b | a．有一个**getHomeHandle()**方法 |
| 2. 远程home接口 | a | b．扩展‘**javax.ejb.EJBObject**’ |
| 3. 本地组件接口 | c | c．方法不能抛出‘**java.rmi.RemoteException**’ |
| 4. 本地home接口 | d | d．可以用于获取一个EJBLocalObject |

**18** 在一个会话bean的生命周期中，哪些方法是容器调用的？（选出所有正确的答案。） （规范：90，97）

- ❑ A. **create()** — 客户调用
- ☑ B. bean 构造函数
- ❑ C. **setRollbackOnly()** — 这些方法由bean调用
- ❑ D. **getUserTransaction()**
- ☑ E. **ejbRemove()**

**19** 关于会话bean类，下面哪些说法是正确的？（选出所有正确的答案。） （规范：95）

- ❑ A. 可以标记为“final”
- ❑ B. 不需要无参数构造函数
- ❑ C. 它们的组件接口方法可以是“private”
- ☑ D. 它们的业务方法名必须以“ejb”开头
- ❑ E. 它们的“ejbCreate”方法不能声明为“final”

**20** 哪种类型的bean可以实现SessionSynchronization接口？ （规范：75）

- ☑ A. 仅有状态会话bean
- ❑ B. 仅无状态会话bean
- ❑ C. 有状态会话bean和无状态会话bean都可以
- ❑ D. 有状态会话bean和无状态会话bean都不可以

**21** 关于无状态会话bean实例，下面哪些说法是正确的？（选出所有正确的答案。）（规范：88）

- ☑ A. 任何实例可以用于任何客户
- ❑ B. 会话状态必须在方法调用之间保留
- ❑ C. 会话状态必须跨事务保留
- ❑ D. 可以钝化
- ❑ E. 不支持事务

# 5 实体bean引言

# 实体都是持久的




**实体bean会一直保存。** 实体bean一直存在。它们是底层持久存储库中某个东西的对象表示（可以把底层持久存储库想成是数据库，因为大多数实体bean表示的就是一个关系数据库中的某个东西）。如果有一个Customer实体bean，那么可能有一个bean表示实体Tyler Durden，ID为#343，另一个bean表示实体Donny Darko，ID是#42。这两个bean分别表示两个实际的实体。实体bean只是持久存储库中已有的、已经存在的某个东西的一种具体实现，而且这些家伙很难被杀死！只要数据还在数据库里，它们就能以实体bean的面貌卷土重来。

# 实体bean

## 标准大纲：

5.1 关于实体bean本地和远程home接口的客户视图，区别哪些说法或例子是正确的，哪些不正确，包括查找实体bean home接口所用的代码，以及提供给客户的home接口方法。

5.2 关于实体bean本地组件接口（EJBLocalObject）和远程组件接口

5.3 （EJBObject）的客户视图，区别哪些说法或例子是正确的，哪些不正确。

5.4 明确以下各类CMP实体bean home方法的使用、语法和行为：查找方法、创建方法、删除方法和home业务方法。

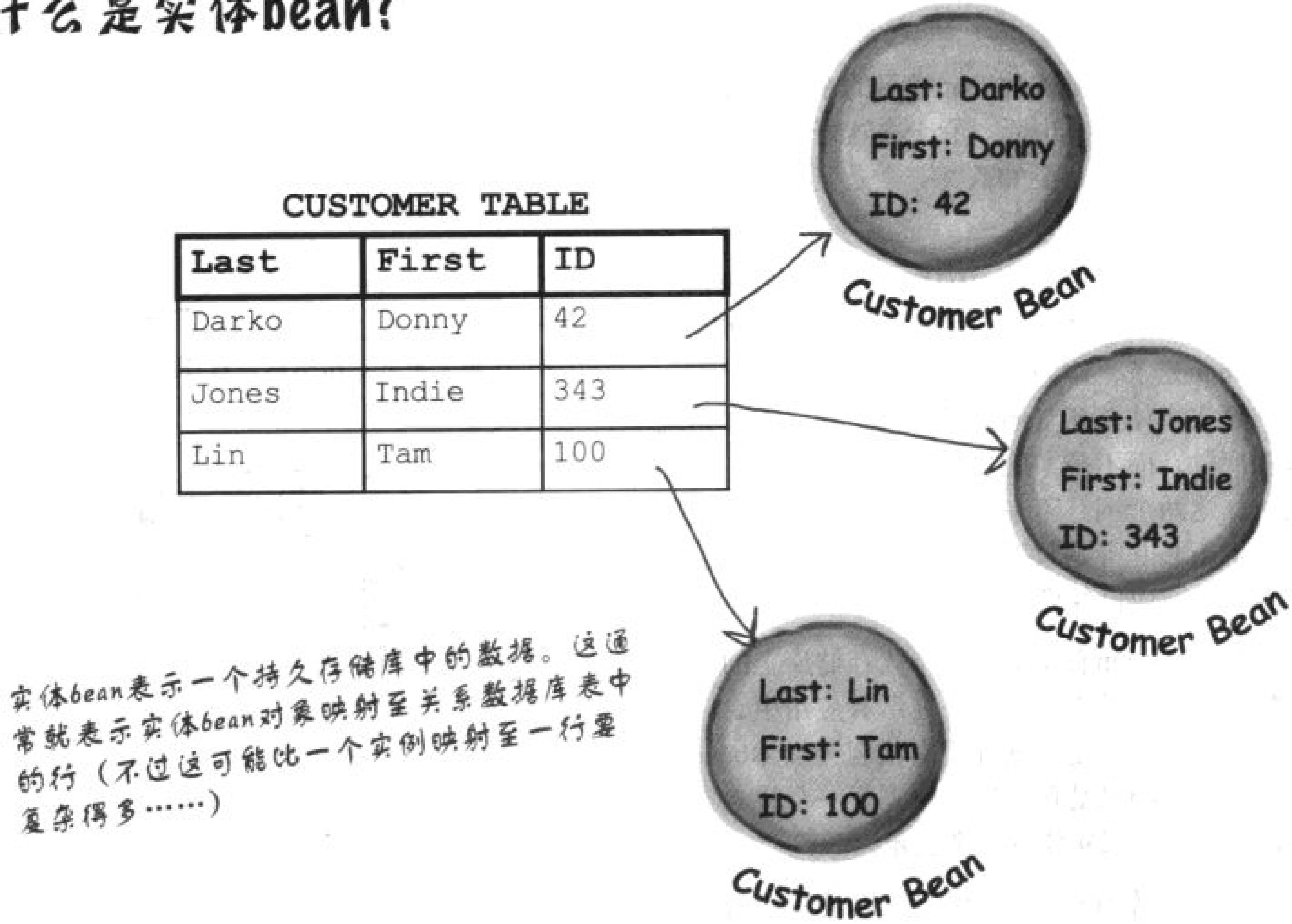
## 具体含义：

必须知道实体bean home接口的规则，特别是这与会话bean的规则有什么不同。你要知道为什么不要求有一个create()方法，但是必须至少有一个查找方法findByPrimaryKey()。你还要知道单实体查找方法和创建方法的返回类型必须是组件接口类型，而多实体查找方法必须返回一个Collection。你要知道如果数据库中没有匹配的实体，单实体查找方法会抛出一个ObjectNotFoundException异常，而多实体查找方法总会返回一个Collection，即使这是一个空集合（数据库中没有匹配的实体时就会返回空集合，而不是抛出异常）。除此以外，你还必须知道home业务方法不要求返回组件接口，这些方法可以返回任何能合法传递的东西。你还要知道，如果从数据库删除一个实体，它就不能再表示为一个实体bean了，但是如果一个实体bean实例死掉（由于出现一个异常，或者服务器崩溃），实体本身还是在持久保存着。只要实体还在数据库里，就能表示为一个实体bean。

一个实体bean组件接口中要如何定义，有关的规则与会话bean组件接口的规则完全一样。不过，从要求5.4可以看出这两种bean的方法有什么不同。

对于实体bean，remove()方法的行为与会话bean删除方法迥然不同，必须了解这里调用remove()实际上意味着“把这个实体从底层持久存储库中删除！”。还必须知道在实体bean上调用create()是指“在数据库中插入一条新的记录行”（不错，从技术上讲，应该是“在底层持久存储库中创建一个新实体”）。你要知道，如果从底层持久存储库删除了一个实体，这个实体bean就会“死掉”，尽管bean实例还会回到池里。

# 什么是实体bean？




实体bean是以一种面向对象的方式来查看持久存储库。规范说的是“持久存储库”，但是没有指定必须是哪一个持久存储库。只要是满足“持久存储”，任何方式都可以，包括关系数据库、对象数据库，甚至像把串行化对象保存到文件这种原始低效的做法也可以。但是在大多数实际情况下，我们说的都是关系数据库。比起一行对应一个bean的情况，对象到关系（object-to-relational，OR）映射可能要复杂得多，但是这一章只讨论简单的情况，到下一章再讨论OR映射的情况。

实体bean是数据对象。它们是……东西，是名词。这与会话bean不同，会话bean是过程，是动词。实体bean可以表示像人、产品、订单、预订、目录、动物等等各种东西。实体bean不能表示像信用卡验证、顾问服务、订单提交、成员注册之类的过程。

当然，几乎在所有设计优良的EJB应用中，实体bean都会与会话bean结合使用，在这些应用中，客户与表示过程的会话bean交互，而会话bean在过程中需要数据时会使用实体bean。

# 实体与实体bean

实体是真实的东西，也就是实体bean表示的东西。反过来却不成立！实体不表示实体bean。看上去这好像没什么，实际上则不然。区别在于，要知道哪一个是真的，哪一个只是真实东西的一个视图。




换句话说，如果你删除了数据库中的真实实体（东西），那么实体bean就会瓦解。噗，灰飞烟灭，什么都没了。不过，如果实体bean死了，作为堆上的一个实例，它并不会把数据库中的真实实体杀死（顺便说一句，从现在开始我们要说“数据库”，而不是“底层持久存储库”，不过，你和我都明白，这两个词并不一定是同义词，而且“底层持久存储库”也不必非得是数据库，尽管一般情况下是这样。这一点我们在前面已经说过了）。

所以，如果你把实体bean看作是放在持久存储库上的一个数据对象，那你的观点就错了。几乎在所有应用中，应用中使用的大多数实体在给定时间并没有实现为实体bean。相反，大多数实体bean只是在应用逻辑需要一个特定实体的时候才会成为这个底层实体的一个表示。

例如，假设你的数据库有10000个顾客。但是在任何给定时刻，客户（通过会话bean中的业务过程）只使用了其中的200个顾客。在这种情况下，可能就只有200个真正表示顾客实体的实体bean。余下的9800个顾客实体只是待在数据库里，等着客户过程什么时候能用到它们。

可以把实体bean想成是演员。在我们前面描述的场景中，你就有200个演员，每个演员都有不同的角色，扮演着数据库中的某个人。但是数据库另外的9800个实体 （也就是其他顾客）却没有人来扮演。

不过，一旦有客户到来，想得到这9800个实体中的一个，比方说，要改变那个顾客的地址，或者检查他的信贷限额，就会选出一个演员（实体bean）来扮演所选顾客的角色。

there are no Dumb Questions

**问：** 能不能这样说……并不是整个数据库都作为实体bean加载进来，对吗？我感觉加载整个数据库太愚蠢了。

**答：** 你是对的，至少前一半是对的。没错，整个数据库确实没有作为实体bean加载进来。可以把实体bean想成是数据库中一部分实体的即时表示，这些实体是客户业务过程中真正需要的。

但这是不是意味着把所有实体都加载进来很愚蠢呢？不一定。当然这要看你的资源需求了，特别是底层数据库中的数据是不是还会增长，还要考虑数据增长得有多快。

尽管在规范中没有要求，但是开发商可以选择让你以某种方式来配置应用，使它能把数据库中的所有实体都预加载为bean。这样一来，最起码地，可以给你一个速度极快的内存中数据库，它会与实际的底层存储库自行同步（稍后你就会了解这一点）。

更进一步，如果你的服务器允许你告诉它：你只能通过实体bean这样惟一一个途径来访问数据库，那么服务器甚至会取消同步，光是维护内存中数据库，而只把它真正需要的东西保存为备份，以备万一服务器崩溃。

实体bean是以一种面向对象的方式来看持久存储库中的数据。

实体是真实的、可以惟一标识的东西，这个东西存在于EJB之外，而实体bean的任务只是成为这个真实、持久实体的一个面向对象视图。

实体bean离开真实实体是没办法存在的，不过真实实体离了bean还能存在。

实体bean不是真实的实体，它只是实体的一个表示或“实现”。

## there are no Dumb Questions

**问：为什么需要实体bean？为什么不直接从会话bean访问数据库？**

**答：** 是可以从一个会话bean直接访问数据库。当然了，确实可以从客户直接访问数据库，不过这样一来，我们不是要退回到原来不可扩展的客户—服务器两层结构吗？而且我们都知道为什么说这种想法不好（不能扩展、业务逻辑放在客户里，很难维护等等）。

如果使用实体bean而不是直接调用数据库，就可以充分利用容器的所有服务，包括能够把多个数据库访问包装在一个事务里。不过，你会看到，使用实体bean最大的好处之一就是容器会自动地完成数据库和实体bean的同步。

不过使用实体bean有一个最有说服力的原因，这就是通过实体bean，你就从原来的关系世界步入对象世界了。换句话说，在你的应用中，你会一直与对象同行，而不用在代码中映射来映射去。另外，如果你在使用CMP（容器托管持久存储），而且一般都会使用CMP（其原因稍后就会清楚了），你就不用在bean代码里放上大堆的SQL。实际上，利用CMP实体bean，你可以假装整个数据库都成为堆上的对象。

当然，也许你对使用JDBC已经很顺手了，但是小组中的其他人呢？需要明确一点，直接以面向对象的方式思考，这比在面向对象和实体代码之间转换更有意义。

**Relax 考试不要求你了解SQL或JDBC。**

在很大程度上是这样。你确实需要知道EJB-QL，这与SQL在某些方面很相似，不过对SQL本身无需任何了解。而且由于JDBC不在考试范围内，所以你必须知道bean要查找一个javax.sql.DataSource，并且可以从这个数据源得到一个java.sql.Connection。在EJB环境一章中会更多地介绍DataSource查找。

# 客户视角看到的实体bean




## 客户想做一些数据库工作，但是要采用一种Java OO方式。

- 建立一个新实体（SQL INSERT）
- 删除一个实体（SQL DELETE）
- 更新实体状态（SQL UPDATE）
- 对实体的查找/查询（SQL SELECT）

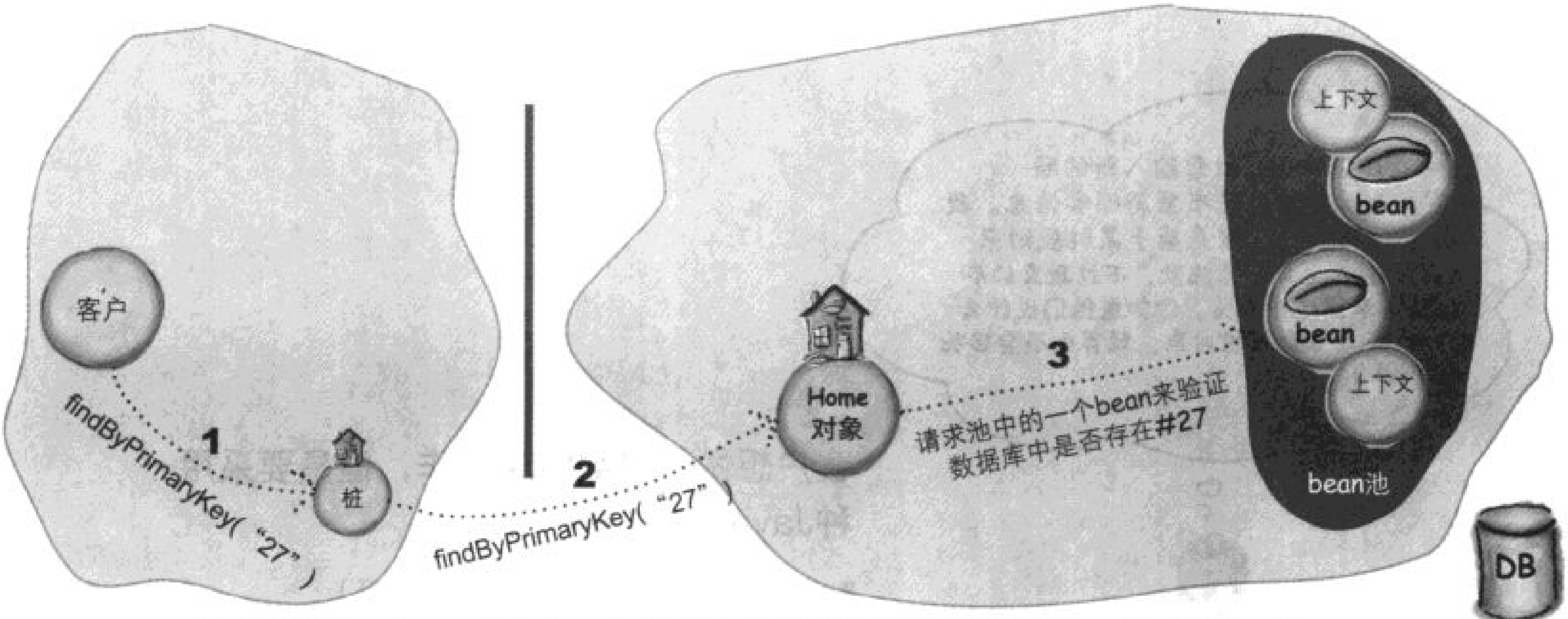
实体bean的客户接口与会话bean的客户接口稍有差别。例如，当一个会话bean客户想得到一个bean（这样它就可以调用这个bean的业务方法），此时客户会调用create()，容器就会分配一个新的EJB对象。

但是如果客户想使用一个现有的实体bean呢？例如，如果客户不想要一个随机的实体，而是想要某个特定实体（比如，Bart Simpson # 12）的EJB对象，该怎么办呢？在这种情况下，create()是不行的。客户不想要一个新实体，它只是想得到一个现有实体的引用。稍后就会看到，实体bean的客户接口会增加（而且必须增加）一个或多个查找方法。

后面两页是有关创建和查找的高层图，即如何创建（插入）实体bean，以及如何找到（选择）实体bean。随着本章的深入，我们还会在这些图的场景中增加更多细节，不过对目前来说，了解这么多就可以了。

# 实体bean概述

### 场景：客户想要一个现有实体的引用

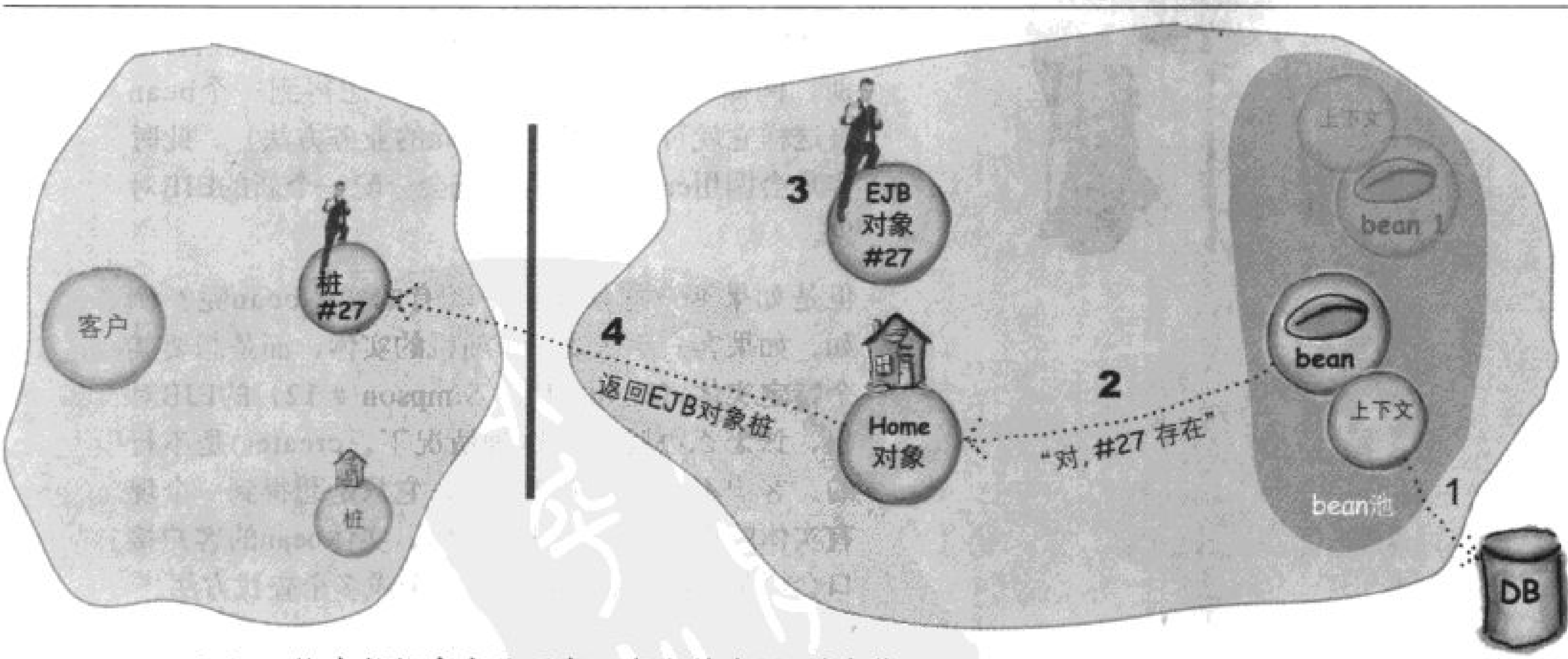



**1** 在实体bean home上完成一个JNDI查找而且得到一个home接口引用之后，客户在home桩上调用findByPrimaryKey( "27" )。

**2** findByPrimaryKey( "27" )方法调用被传递给home对象。

**3** 容器请求池中的一个bean来验证数据库中是否存在#27。

---




**1** bean检查数据库中是否有一个主键为#27的实体。

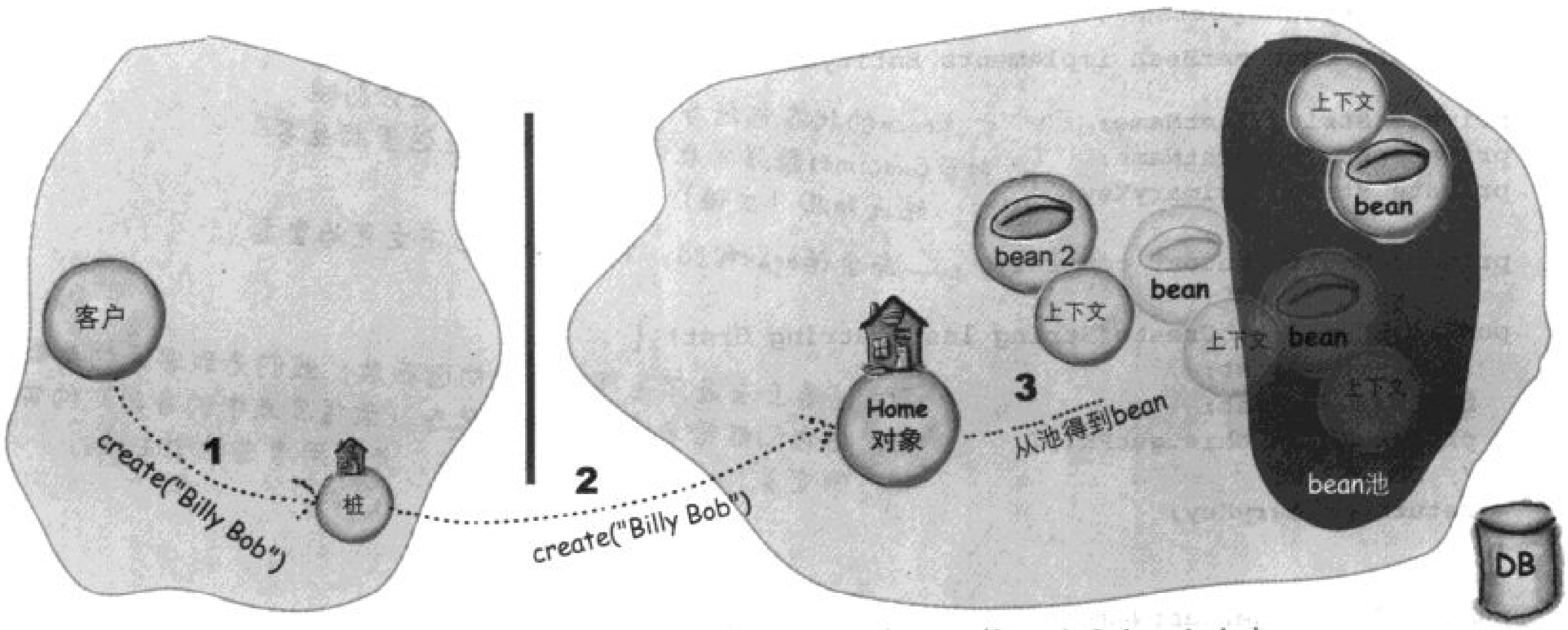
**2** bean告诉home，数据库里有#27。

**3** 容器为#27建立或查找一个EJB对象（可能已经有一个这样的对象了）。

**4** 容器返回对应#27的桩。

## 实体bean概述

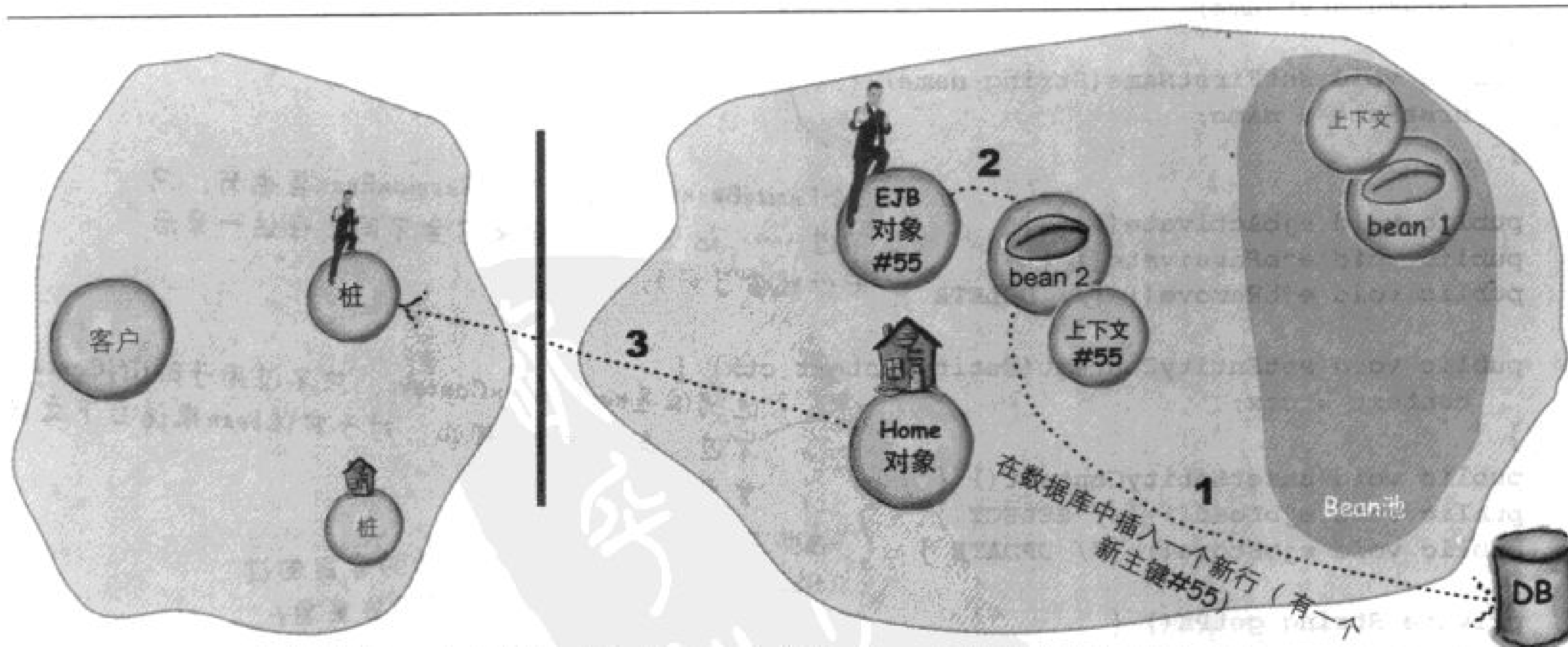
### 场景：客户想创建一个新实体




**1** 在实体bean home上完成JNDI查找并且得到一个home接口引用后，客户在home桩上调用create（“Billy Bob”）。

**2** create（“Billy Bob”）方法调用被传递给home对象。

**3** 容器从这种类型bean的池中拉出一个bean（可能是池中任何随机的bean）。要注意这个bean已经有自己的上下文。




**1** 容器（或bean）在数据库中插入一个新行，bean生成一个新主键。

**2** bean连接到一个EJB对象，上下文和EJB对象都得到新的主键。

**3** 容器返回新创建实体（Billy Bob, #55）的一个桩。如果客户惟一的目标就是插入这个新行，那么客户可能甚至不关心返回的桩。

# 一个简单的Customer实体bean

```
// 这里是package和imports语句
public class CustomerBean implements EntityBean {

    private String lastName;
    private String firstName;
    private String primaryKey;

    private EntityContext context;

    public String ejbCreate(String last, String first) {
      lastName = last;
      firstName = first;
      primaryKey = this.getPK();
      // DB INSERT
      return primaryKey;
    }

    public String getLastName() {
       return lastName;
    }
    public void setLastName(String name) {
       lastName = name;
    }
    public String getFirstName() {
       return firstName;
    }
    public void setFirstName(String name) {
       firstName = name;
    }

    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() {// DELETE }

    public void setEntityContext(EntityContext ctx) {
       context = ctx;
    }
    public void unsetEntityContext() { }
    public void ejbLoad() {// SELECT }
    public void ejbStore() {// UPDATE }

    private String getPK() {
      return ""+ (int) (Math.random() * 42);
    }

    public String ejbFindPrimaryKey(String pk) {// SELECT, 返回 pk }
    public Collection ejbFindByCity(String city) {// SELECT, 返回键的集合 }
}
```

实现EntityBean接口（而不是SessionBean）

实体bean的状态包括各个字段，这些字段分别映射至Customer数据库表中的各个列，在这里就是名字、姓氏和ID（主键）

与会话bean相比，实体bean的上下文更为重要

这看上去是不是很像一个构造函数？我们先取客户的参数，把它们赋给表示实体持久状态（数据库表中的各列）的实例变量。ejbCreate()返回主键，先不用考虑这个主键。

这些是普通的Java获取方法和设置方法，用于获取和设置持久字段。稍后可以看到，这里的神奇之处在于这些设置方法的结果最终会导致一个数据库更新！

3个EntityBean的容器回调，SessionBean里也有。只不过…… 这里它们的含义完全不同，在这一章后面你就会了解到。

这就像是setSessionContext，只不过用于EntityContext。不过，与会话bean相比，对于实体bean来说上下文更要重要。

这些是EntityBean接口中3个新的容器回调（SessionBean中没有）。它们相当重要！

世界上最糟糕的主键算法。通常我们要使用客户提供的信息，或者可能使用数据库的自动生成的键。

（现在先忽略最后这两个方法）

# 实体bean的客户视图

我们的Customer bean还没有完成，所以不要太依赖它！而且实际上，这个代码是bean托管持久存储（bean-managed persistence，BMP）的一个例子，但在这本书中不会使用BMP。我们对BMP几乎没有谈什么，而是会在这一章和下一章花大量笔墨来介绍容器托管持久存储（container-managed persistence，CMP），相比之下CMP更常用。不过，现在我们强调的是实体bean的客户视图，这个简单的bean是一个起点，只是让你看看实体bean代码。

假设一个Customer bean表示底层数据库中的一个Customer实体（即一个真实的顾客），这个实体bean应该有哪些行为？换句话说，客户希望利用一个Customer或多个Customer能做些什么事情？

任何处理数据库记录的事情都能做！我们前面提到过，这包括：建立一个新Customer、删除一个Customer、更新一个Customer的字段（列），在Customer数据库上查询/查找。

**BRAIN POWER**

考虑以下操作，看看对于各个操作，哪一个客户接口（组件接口还是home接口）更合适。要记住，实体bean的规则可能与会话bean接口的规则不同。如果你认为两个接口都合适，那就都打上勾（我们已经得出了第一个操作的适合接口）。

| home | 组件 | 操作 |
|---|---|---|
| ✔ | | **建立一个新顾客** |
| | | **修改一个现有顾客的电话号码** |
| | | **查找所有住在Pleasantville的顾客** |
| | | **删除所有先前声明为“不活动”的顾客** |
| | | **删除一个特定顾客** |
| | | **得到一个特定顾客的街道地址** |

# 实体bean远程组件接口

实体bean的组件接口与会话bean的组件接口很类似，它也有业务方法，它也要扩展javax.ejb.EJBObject。这说明，客户不光可以看到你在组件接口中声明的方法，还可以看到EJBObject的方法（getHandle()、remove()等等）。

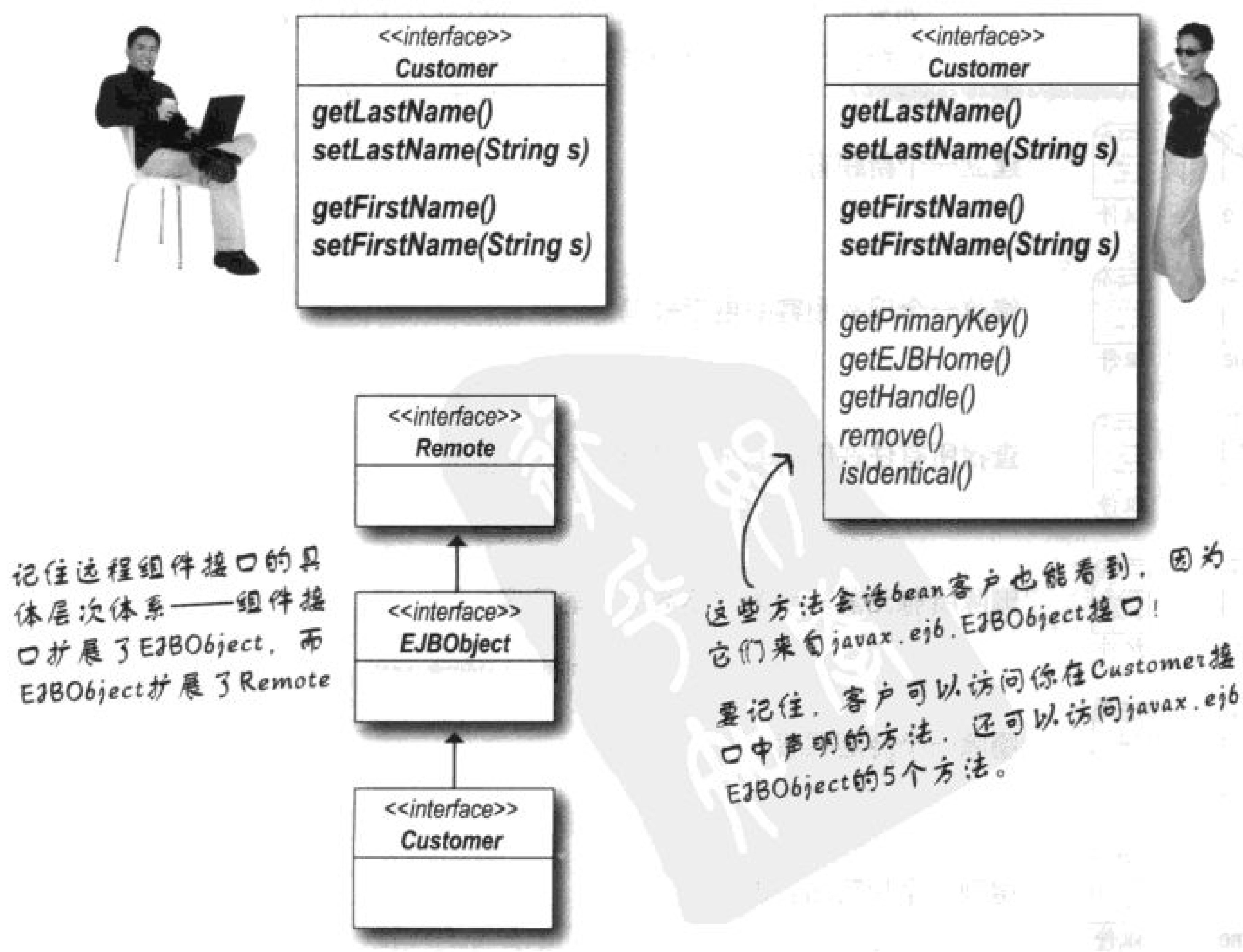
但是哪些业务方法放到组件接口里呢？

**这些方法与一个实体有关！**

客户有一个实体bean的引用时（当然，这表示bean的EJB对象的引用），客户就有了一个特定实体的引用。Fred Flintstone, #999； Marge Simpson； #728；Roy Rodgers, #1957；等等。

那么客户想对一个引用做些什么呢？比如说，对Marge Simpson？删除她、修改她的姓、得到她的句柄，还是得到她的home（这样客户就可以得到其他顾客的引用了）。不过，要知道，这个简单的Customer bean并没有什么用处，它只有得到或设置Customer名的一些方法。后面我们还会完整地进行构建。

**你写的：**　　　　**客户看到的：**



# 实体bean远程组件接口

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;

public interface Customer extends EJBObject {

    public String getLastName() throws RemoteException;
    public void setLastName(String lastName) throws RemoteException;

    public String getFirstName() throws RemoteException;
    public void setFirstName(String firstName) throws RemoteException;

}
```

## 远程组件接口规则

1. 导入javax.ejb.*和java.rmi.RemoteException

2. 扩展javax.ejb.EJBObject

3. 声明一个或多个业务方法，业务方法都要抛出一个RemoteException

   - 参数和返回类型都必须是RMI-IIOP兼容的（Serializable、基本类型、Remote，或上述类型组成的数组或集合），

   - 可以有重载方法

   - 每个方法必须声明一个RemoteException

   - 可以声明你自己的应用异常，但是不能是运行时异常（换句话说，必须是编译器检查的异常，即Exception的子类，而不是RuntimeException的子类）

   - 方法的名字可以任意，只要不以“ejb”开头就行

# 实体bean远程home接口

## 它与会话bean home接口有什么区别：

**① 一般更需要查找一个现有的实体而不是创建一个新实体**

作为客户，你很可能会花大量的时间使用现有顾客的引用，而不是创建新的顾客。不论你是在更新一个特定顾客，还是对多个顾客完成一个批操作，都需要home接口代码来选择顾客，而不是需要创建方法。实际上，对实体bean来说，create()方法甚至是可选的！（这可能出于某种原因，例如，也许你的策略是：数据库中要增加新的项，必须直接通过一个数据库管理工具来完成。）不过，你必须为实体bean home至少放一个查找（finder）方法，查找方法有多少，这一点随你，不过每个实体home中都必须有一个findByPrimaryKey(String primaryKey)（这说明它可能是home接口中声明的惟一方法）。

**② 你可能想完成涉及多个实体的查询**

由于会话bean表示过程，所以如果说得到同一个过程的多个实例，这是没有什么意义的。但对于实体bean，则可能希望像在数据库表上一样，把同样的事情做好多次，比如找出所有住在Helsinki而且喜欢冲浪的顾客。

### 你写的：



```
<<interface>>
CustomerHome
-----------------------------------
create(String last, String first)
findByPrimaryKey(String key)
findByCity(String city)
```

### 客户看到的：

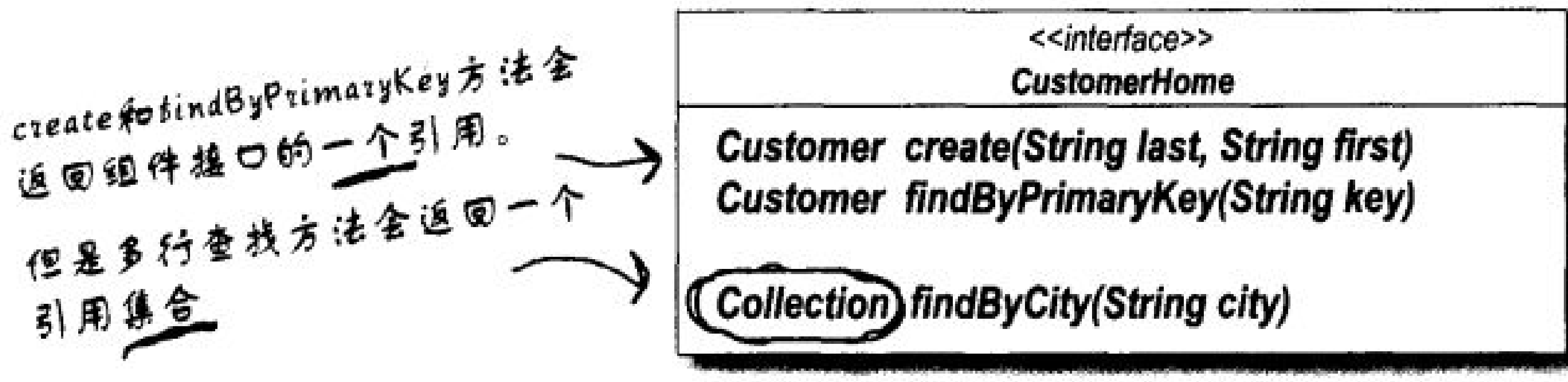
```
<<interface>>
CustomerHome
-----------------------------------
create(String last, String first)
findByPrimaryKey(String key)
findByCity(String city)

getEJBMetaData()
getHomeHandle()
remove(Handle h)
remove(Object key)
```



实体bean客户可以像会话bean客户一样看到同样的4个方法，因为会话bean和实体bean home接口都必须扩展javax.ejb.EJBHome。对于实体bean，两个remove()方法都能用，而会话bean只能使用取句柄（Handle）的删除方法。

# 客户到底想从一个实体bean home得到什么？




对于会话bean，这个问题的答案很简单，就是组件接口的一个引用。而这也正是create()方法必须返回的东西。

对于实体bean，create()方法也是一样的，它们必须返回组件接口的一个引用，在这里就是刚创建实体的组件接口。

但是，如果你想查找一个现有的实体bean，而不是建立一个新实体bean，又会怎么样呢？实体bean有一个必不可少的findByPrimaryKey()方法，要查找现有的实体，这正是findByPrimaryKey()方法要做的，如果有与所查主键匹配的bean，findByPrimaryKey()方法必须返回这个bean组件接口的一个引用。

但是如果没有匹配的实体该怎么办呢？如果带这个键的实体在数据库中不存在，客户会得到一个javax.ejb ObjectNotFoundException异常。所以，findByPrimaryKey()方法的返回类型总是与create()的返回类型相同，都是这种类型bean的组件接口（当然，会话bean客户接口的规则在这里同样适用，远程home接口必须返回远程组件接口，而本地home接口必须返回本地组件接口）。

还有一个方法不能返回组件接口：这就是多实体查找方法，比如findByCity()方法。对于多实体查找方法，客户的目标并没有改变，还是想得到组件接口的一个引用，只不过这一次可能是多个引用的一个集合。每个引用对应方法参数所指定城市中的各个顾客实体。

注意：如果多实体查找方法没有找到任何匹配，客户不会得到异常！相反，客户还是会得到一个集合，不过这只是个空集合，也就是没有任何元素的Collection。在任何实体都与查找规则不匹配的情况下，只有单实体查找方法才会抛出异常。

**实体bean home中的创建和查找方法总是返回bean的组件接口。**

**对于create()和findByPrimaryKey()，客户会得到一个EJB对象的引用。**

**对于多实体查找方法，客户可能会得到一堆EJB对象引用，每个引用分别对应与查询匹配的各个bean。**

# 实体bean远程home接口

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;
import java.util.Collection;

public interface CustomerHome extends EJBHome {

    public Customer create(String last, String first) throws CreateException, RemoteException;

    public Customer findByPrimaryKey(String key) throws FinderException, RemoteException;

    public Collection findByCity(String city) throws FinderException, RemoteException;
}
```

# 查找方法也有不好的一面……




对于一个远程接口，查找方法和创建方法会返回远程桩。

这说明必须对每个远程桩做远程方法调用来得到你想要的数据。




## there are no Dumb Questions

**问：** **等等……我想，无论如何，全部由客户做这些方法调用不是一个好的设计吧。难道不能让客户总是通过一个会话bean，再由这个会话bean与实体bean对话吗？**

**答：** 差不多。你想的正是会话外观（Session Facade）J2EE设计模式。不过，如果你确实在实体bean的前面放上一个会话bean，这个会话bean还是一个客户。也许它的效率会高得多，因为会话bean在网络上的位置可能没那么远（甚至没准与实体在同一个服务器上）。但是如果你坚持位置独立性，那你的会话bean还是会使用实体bean的远程接口，所以这里仍然会有很大的开销。

**问：** **组件接口中难道不能有一个返回数据的业务方法吗？比如，返回一个String集合？**

**答：** 你越来越接近了。不错，在EJB 1.1里正是这样做的。不过，这还是不算太好，因为你必须先得到某个顾客的引用，这样才能请求那个顾客返回所有顾客的数据。

## 解决之道：home业务方法

不错……在讨论实体bean时，业务方法不只是针对组件接口。对于EJB 2.0，实体bean home可以有（注意了）不必返回组件接口的方法！home业务方法可以返回任何东西（当然，还有一个限制，这就是远程home方法返回的值必须是RMI-IIOP兼容的）。

home业务方法非常适合批操作，另外对于某些查询方法，如果客户不需要（或者不想要）EJB对象引用，只是想得到实体的数据（换句话说，就是实体的一个或多个持久字段的数据），也很适合采用home业务方法。例如，可以在Customer bean里放上一个home业务方法，比如getAllCustomerInfo()，由它返回一个String集合，其中包括构成顾客信息的所有数据。还有更好的，可以让这个方法发回一个CustomerInfo对象集合，这里CustomerInfo是一个类，其中只存放对应Customer持久状态的数据（和获取方法）。这样一来，客户就可以完成本地调用，从CustomerInfo对象得到他需要的数据，而不必在组件接口上做远程调用。

CustomerInfo类是值对象（Value Object）类的一个例子，所谓值对象，实际上就是带获取方法（表示实体的持久字段）的一个类（也可能带设置方法，这取决于设计）。而值对象同样有不好的一面，一旦创建了值对象，从这一刻起，数据就开始变得过时。

当然我们现在也可以直接解释给你听，但是这样一来，你就不会动脑筋去想，这就失去了一个锻炼的大好机会。所以，你怎么不自己想想看，为什么发回CustomerInfo对象（这样客户就能轻松地询问这些对象，也就是在这些对象上调用方法）也有不好的一面。我们会大量使用值对象，但是到底使用一个返回EJB对象引用（特别是远程引用）的home查找方法，还是使用返回值对象的home业务方法，在二者间做选择时一定要考虑到这种权衡。

**Home业务方法可以返回不是EJB对象引用的任何东西！如果客户只想要实体数据，而不想要实体本身的引用，这种查询就很适合使用home业务方法。**

**home业务方法对批操作也很合适，或者如果你想使用多个特定实体，又不想返回组件接口的引用，此时也可以采用home业务方法。**

## 远程home接口的规则

1. 导入javax.ejb.*和java.rmi.RemoteException

2. 扩展javax.ejb.EJBHome

3. 声明（可选）一个或多个create()方法，这些创建方法必须返回远程组件接口，而且要声明RemoteException和CreateException。每个create()方法都必须以“create”前缀开头

4. 声明findByPrimaryKey()方法，它必须返回远程组件接口，而且要声明一个RemoteException和一个FinderException

5. 声明（可选）一个或多个其他的查找方法，这些查找方法必须返回远程组件接口（单实体查找方法），或者必须返回java.util.Collection（多实体查找方法）。所有查找方法都必须声明一个RemoteException和一个FinderException

6. 声明一个或多个home业务方法

   - 参数和返回类型必须是RMI-IIOP兼容的（Serializable、基本类型、Remote，或上述类型组成的数组或集合）
   - 可以有重载方法。
   - 每个方法必须声明一个RemoteException。
   - 可以声明你自己的应用异常，不过这些异常不能是运行时异常（换句话说，它们必须是编译器检查的异常，也就是Exception的子类，而不是RuntimeException的子类）
   - 方法名可以任意，只要不以“create”、“find”或“remove”开头就行

## Sharpen your pencil

针对客户想对一个实体bean做的4种数据库操作（SQL命令），列出bean接口中哪些方法与这些数据库操作有关。你不用了解SQL，但是必须理解这4个数据库操作的含义，而且要知道它们与bean类中的方法如何对应。

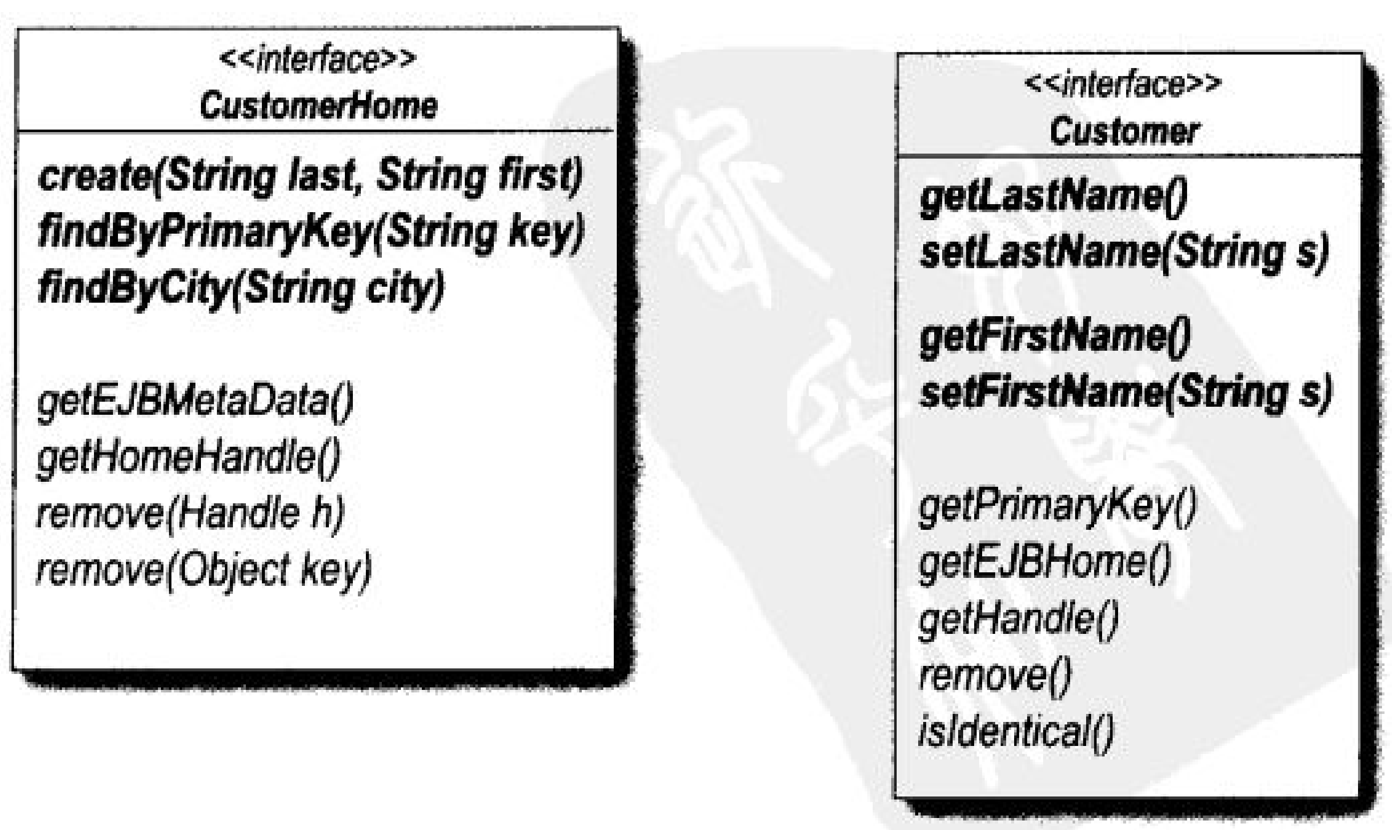
根据接口中的方法列表，填写对应各数据库操作的方法。

**INSERT:**

**DELETE:**

**UPDATE:**

**SELECT:**



# 会话bean create() 与实体bean create()

## ① 有状态会话bean create()

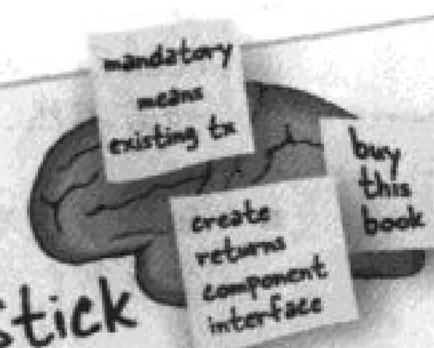
- 客户调用这个方法得到一个新的有状态会话bean（专用于这个客户）的EJB对象引用。
- 这个方法可以有（通常都有）参数，bean使用这些参数来完成特定于客户的初始化（在运行任何业务方法之前）。
- 客户调用create()时，容器建立一个新的会话bean。

## ② 无状态会话bean create()

- 客户调用这个方法得到一个bean的EJB对象引用 。
- 这个方法没有参数，而且bean不做任何特定于客户的初始化（因为调用bean的ejbCreate()时，这个bean并没有与任何客户关联!)。
- 客户调用create()时，容器不建立一个新的会话bean，而且不会从池中拉出一个bean，直到客户调用业务方法时才会真正从池中取出一个bean。

## ③ 实体bean create()

- 客户调用这个方法在数据库中插入一个新行！* 不过对客户来说，最终结果还是一个EJB对象引用（在这种情况下，这是新创建实体的EJB对象引用）。
- 一般都是有参数的（尽管对此没有要求，不过很难想像一个没有参数的create()，比如“嘿，数据库，创建一个新顾客……不，我不知道他的名字，也不知道ID，什么也不知道……给我建一个就得了”）。
- 容器不会建立一个新的实体bean，而是会从池中取出一个来运行ejbCreate()方法。要记住，ejbCreate()方法必须取create()的参数，并以某种方法在底层持久存储库中创建一个新实体（或者至少支持容器创建一个新实体）。




**Make it Stick**

每朵鲜花，
都会带来快乐，
每个创建方法，
都会建立新行。

记住……
实体bean: 创建方法 == 新行, 新EJB对象
有状态bean: 创建方法 == 新bean, 新EJB对象
无状态bean: 创建方法 == 新EJB对象

# 会话bean remove()与实体bean remove()

## 有状态会话bean create()

- 客户调用这个方法告诉容器他已经用完了这个bean。
- 容器调用bean的ejbRemove()（除非bean已经钝化），并杀死这个bean（把它想成扔到垃圾堆的食物）。
- 删除了这个bean之后，如果客户试图再使用EJB对象引用，会得到一个异常。

## 无状态会话bean create()

- 客户调用这个方法告诉容器他已经用完了这个bean。
- 容器得到调用，并说："我管你呢！你真地认为你那么重要吗？这个bean已经回到池里了。"容器不会调用bean的ejbRemove()。想想看，如若不然，它该调用哪个bean的ejbRemove()呢？
- 删除了这个bean之后，如果客户试图再使用EJB对象引用，会得到一个异常。

## 实体bean create()

- 客户调用这个方法通知容器删除带这个主键的实体。
- 容器调用bean的ejbRemove()方法，如果bean支持客户触发的删除，就会从底层持久存储库删除这个实体。换句话说，数据库中的这一行会成为历史，噗，灰飞烟灭，没有了。
- 删除了这个bean之后，如果客户试图再使用EJB对象引用，会得到一个异常。
- 实际上，任何客户都不能使用该实体的EJB对象引用了！




会话bean上的remove()意味着<u>客户</u>用完了这个bean。

实体bean上的remove()意味着<u>所有人</u>都用完了这个bean！







停！
导演
pirates of the CamBEAN


看来有人没有好好看剧本……要知道本来不该是这样……不是bean实例被杀死，该死的是数据库中的实体。客户调用 remove()的时候，你应该回到bean池里才对。
导演
pirates of the CamBEAN
但是我还没演过死的一幕呢。我知道，只是数据库里的实体被删除，但我只是想来点更震撼一点的。我明白，如果我抛出一个非受查异常就会死掉，但是这好像永远也不会发生一样……我一直期待着有人能在我的墓碑旁念着“我们尊敬的bean……”

# 实体/bean/实例的死亡

我们都知道了，实体bean是底层持久存储库中某个真实实体（通常就是数据库中的一行）的一个表示。但是，关于实体、实体bean还有实体bean实例之间有什么区别，还是很容易混淆。

### 实体

这是底层持久存储库里的真实东西，数据库中的行（不过可能比这更复杂）。如果从底层库中删除了一个实体的行，不管是通过一个直接的数据库删除（比如，有人使用一个数据库管理工具来删除），还是由于有人在bean的home或组件接口上调用了remove()，这个实体就会死掉。

### 实体*bean*

这是表示底层真实实体的组件。不过，这里有一点麻烦……它是类吗？它是接口吗？还是bean类的实例呢？在开发和部署时，实体bean是整个组件（包括两个接口、部署描述文件和bean类）。但是到了运行时，就不那么清楚了。有时，我们谈到“实体bean”时是指：可以把一个特定实体表示为一个bean。换句话说，如果数据库中有一个对应Bo Rodgers的实体，我们就可以说有一个Bo Rodgers实体bean，即使当前并没有表示这个实体的bean实例！如果存在对应一种特定类型bean的实体（如Customer Fred Foo），则称该实体的实体bean存在。如果底层实体被删除，则称相应的实体bean死亡，比如“没有Fred Foo实体bean”。但是……这并不是说堆上的实例死掉了。

### 实体*bean*实例

**这是堆上的bean类实例。**bean的死亡与数据库紧密相关，不过bean实例的死亡（比如“你要被垃圾回收了”）则受控于容器（或服务器）是否崩溃。

对，这里是有一点容易混淆。你必须知道在哪一个上下文里用“bean”这个词。如果它是指“表示实体的bean”，那么该实体死亡时这个bean就会死掉，而且这个实体的EJB对象也就没有了。不过（正是这里有麻烦），此时实体bean实例却不会死；它只是回到池里。要把“实体bean”想成是概念性的，而不是具体的。在大多数情况下，都不用区别bean和bean实例，或者它们的区别很明显，因此不算是个问题。

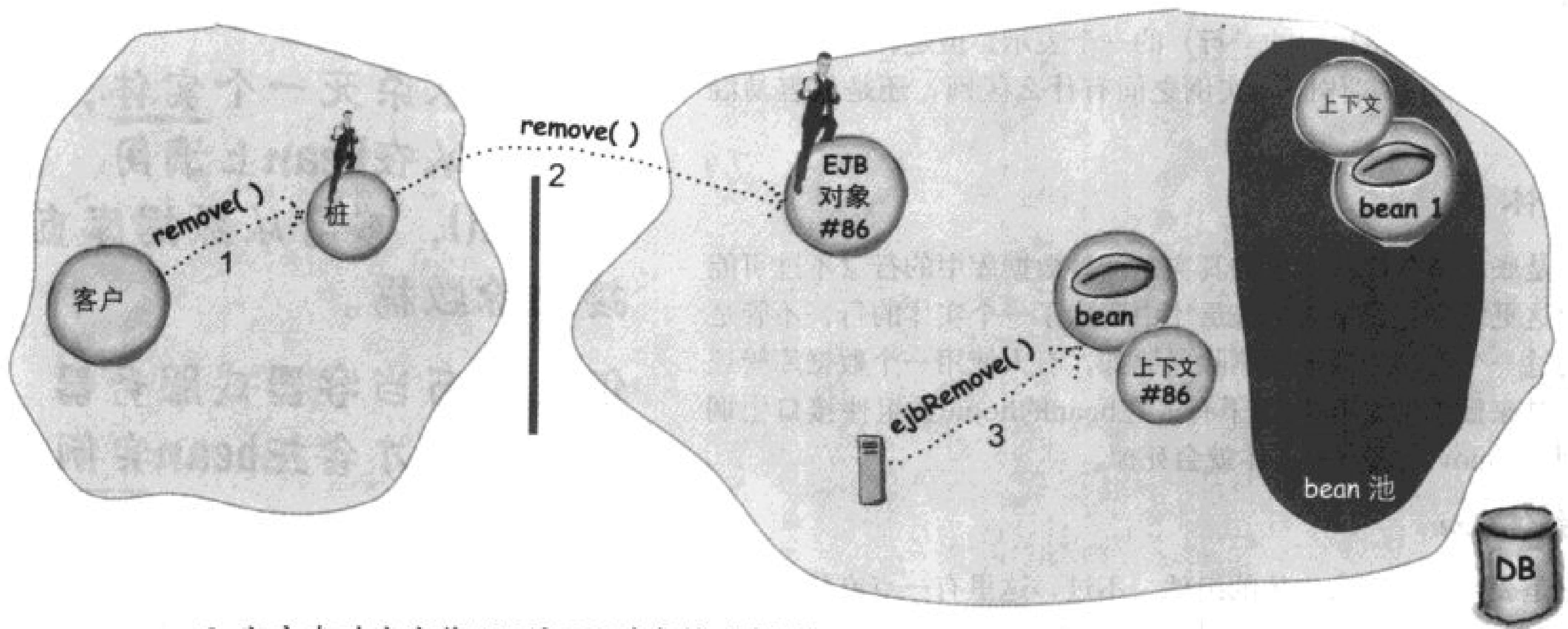
**客户可以杀死一个实体，为此可以在bean上调用remove()，或者从数据库直接删除数据。**

**但是只有当容器或服务器崩溃时，才会把bean实例杀死。**

**如果我们说“实体bean #42被杀死”，底层实体就没了，而且#42的EJB对象也没了，不过扮演#42的bean实例还活着。**



## 客户调用remove()

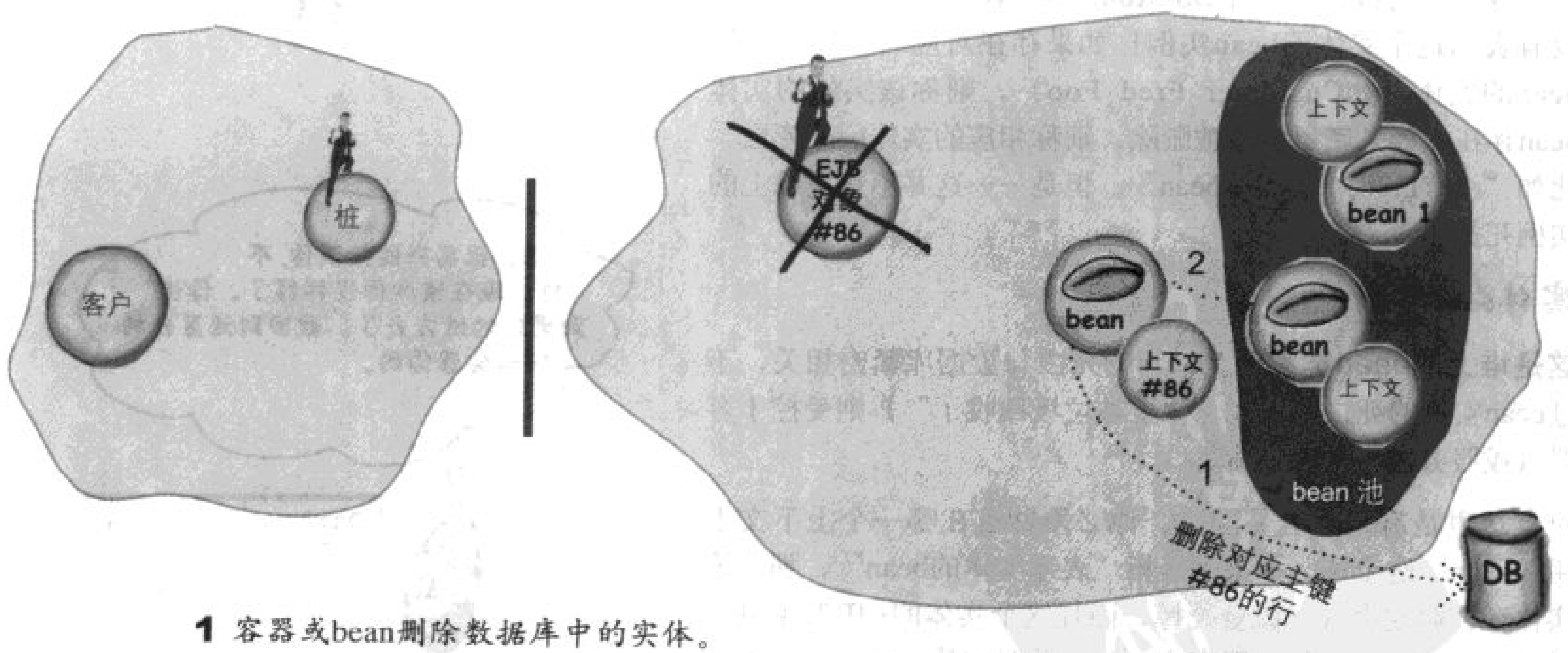



1 客户在对应实体#86的EJB对象桩上调用remove()。

2 remove()方法调用被传递给EJB对象。

3 容器在bean上调用ejbRemove()。

---




1 容器或bean删除数据库中的实体。

2 bean没有了身份（也就是说，它不再表示实体#86），并回到池里去。与此同时，对应#86的EJB对象被删除，所以如果客户又使用它来调用一个方法，客户的桩就会抛出一个异常。

# 实体bean客户视图

## 要点

- 实体是持久存储库中的一个真实的东西，存在于EJB之外，实体bean是一个实体的面向对象表示或实现。
- 客户使用实体bean以一种面向对象方式来完成数据库操作。数据库操作包括：创建新实体（数据库插入）、删除实体（数据库删除）、更新实体状态（数据库更新）和查找实体（数据库选择）。
- 实体bean远程组件接口扩展了EJBObject。会话bean远程组件接口同样扩展了这个接口，这个接口针对会话bean和实体bean并没有不同的版本。这说明，客户会看到组件接口里的所有方法，外加EJBObject里的另外5个方法。
- 实体bean组件接口通常包含一些字段值的获取方法和设置方法，这些字段值对应于数据库的列，如getLastName()、getHomePhone()、setFirstName()等等。
- 实体bean组件接口方法通常要由一个特定的、可惟一标识的实体运行。例如，在一个主键为#420的实体上调用getLastName()，会返回数据库中主键为#420的实体的姓，Dan Doof。
- 如何编写实体bean远程组件接口，有关的规则与会话bean的相应规则完全相同，包括：要扩展EJBObject，所有方法都要声明RemoteException，只能使用RMI-IIOP类型作为参数和返回类型，方法名不能以前缀“ejb”开头，等等。
- 实体bean home接口与会话bean的home接口完全不同，因为通常要查找实体bean，而不是创建实体bean。换句话说，客户很可能要访问一个现有的实体，而不是建立一个新实体（这意味着数据库中的一个新行）。
- 在一个实体home中，不一定非要有一个create()方法（因为实体bean中的create()方法用于向数据库中插入新实体，而你不一定允许你的客户这样做）。
- 实体bean home接口可以有单行或多行查找方法。创建方法和查找方法都会返回一个bean的组件接口，不过多实体查找方法会返回组件接口引用的一个Collection。
- 每个bean home都必须至少有一个方法，这就是findByPrimaryKey()方法，这个方法要查找一个特定的实体，并返回这个实体的组件接口（也就是该实体EJB对象的一个引用），或者抛出一个异常（如果没有找到匹配的实体）。
- 如果没有找到匹配的实体，多实体查找方法不会抛出异常，而只是返回一个空集合。
- 实体home接口还可以有home业务方法，用于完成应用于多个实体（而不是应用于一个特定实体）的操作。批更新也很适合使用home业务方法。
- home业务方法真正的好处在于，不同于创建方法和查找方法，home业务方法返回的可以不是EJB对象引用。如果客户只想要数据，比如，只想要表示各顾客姓名和电话号码的String Collection，home业务方法就可以做到，查找方法则不行。
- 实体bean create()与会话bean create()有很大不同，因为实体bean create()会向底层持久存储库插入一个新实体（也就是数据库中的新行）。
- 实体bean remove()与会话bean remove()截然不同!客户在一个实体bean上调用remove()时，它是要从数据库删除这个实体！这说明所有人都用完了这个bean。

# COFFEE CRAM
# 模拟测验

**1** 关于如何得到一个实体bean的home接口，下面的哪个说法正确？

- ❑ A. 应当对本地home接口使用**`narrow()`**方法
- ❑ B. 应当对远程home接口使用**`narrow()`**方法
- ❑ C. 应当对本地和远程home接口都使用**`narrow()`**方法
- ❑ D. 本地和远程home接口都不使用**`narrow()`**方法

**2** 实体bean远程组件接口中可以有哪些功能？（选出所有正确的答案。）

- ❑ A. 创建新的实体对象
- ❑ B. 查找现有的实体对象
- ❑ C. 删除现有的实体对象
- ❑ D. 执行home业务方法
- ❑ E. 获取EJBMetaData接口

**3** 客户可以通过什么办法得到一个现有实体对象本地组件接口的一个引用？（选出所有正确的答案。）

- ❑ A. 调用 **`ejbCreate()`**
- ❑ B. 调用 **`getSessionContext()`**
- ❑ C. 由句柄得到引用
- ❑ D. 作为方法调用的一个参数接收引用
- ❑ E. 使用本地home接口中定义的一个查找方法

**4** 实体bean的本地home接口中可以定义多少个创建方法？

- ❏ A. 0
- ❏ B. 只有1个
- ❏ C. 0或1个
- ❏ D. 0或多个
- ❏ E. 1或多个

**5** 客户可以通过什么办法得到一个现有实体对象远程组件接口的一个引用？（选出所有正确的答案。）

- ❏ A. 调用 **`ejbCreate()`**
- ❏ B. 由句柄得到引用
- ❏ C. 根据实体对象的主键调用一个方法
- ❏ D. 作为一个方法调用的参数接收引用
- ❏ E. 使用在远程home接口中定义的一个查找方法

**6** 可以对主键类使用什么方法来确定两个键指向同一个实体？（选出所有正确的答案。）

- ❏ A. 使用 == 操作符
- ❏ B. 使用**`equals()`**方法
- ❏ C. 使用**`isIdentical()`**方法
- ❏ D. 以上都不行

**7** 下面哪些方法可以确定两个实体EJB对象引用指向同样的实体对象？（选出所有正确的答案。）

- ❏ A. 使用 == 操作符
- ❏ B. 使用**`equals()`**方法
- ❏ C. 使用**`isIdentical()`**方法
- ❏ D. 以上都不行

**8** 关于实体bean远程组件接口的客户视图，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 多个客户可以并发地访问同一个实体对象
- ❑ B. 可以使用这个接口中的一个方法创建新的实体bean
- ❑ C. 容器崩溃后，实体bean可能不能幸存
- ❑ D. 业务方法不能返回实体对象的一个引用

**9** 实体bean本地home接口中可以声明多少个查找方法？

- ❑ A. 0
- ❑ B. 只有1个
- ❑ C. 0或1个
- ❑ D. 0或多个
- ❑ E. 1或多个

**10** 对于一个本地home接口的 create()方法，以下哪个声明是合法的？（选出所有正确的答案。）

- ❑ A. `public Cust create(int x);`
- ❑ B. `public void create(int x) throws CreateException;`
- ❑ C. `public Cust create(int x) throws CreateException;`
- ❑ D. `public Cust create(int x) throws CreateException, RemoteException;`

**11** 对于一个实体bean home业务方法，以下哪个方法名是合法的？（选出所有正确的答案。）

- ❑ A. create
- ❑ B. createCust
- ❑ C. removeAll
- ❑ D. findCust
- ❑ E. selectCust

# COFFEE CRAM 模拟测验答案

**1** 关于如何得到一个实体bean的home接口，下面的哪个说法正确？ （规范：110）

❑ A. 应当对本地home接口使用**narrow()**方法

☑ B. 应当对远程home接口使用**narrow()** 方法

❑ C. 应当对本地和远程home接口都使用**narrow()** 方法

❑ D. 本地和远程home接口都不使用**narrow()** 方法

**2** 实体bean远程组件接口中可以有哪些功能？ （选出所有正确的答案。）

❑ A. 创建新的实体对象 — home方法

❑ B. 查找现有的实体对象

☑ C. 删除现有的实体对象

❑ D. 执行home业务方法 — home方法

❑ E. 获取EJBMetaData接口

**3** 客户可以通过什么办法得到一个现有实体对象本地组件接口的一个引用？ （选出所有正确的答案。） （规范：119）

❑ A. 调用 **ejbCreate()** — 我们要的是现有的实体:)

❑ B. 调用 **getSessionContext()**

❑ C. 由句柄得到引用 — 句柄对应的是远程接口

☑ D. 作为方法调用的一个参数接收引用

☑ E. 使用本地home接口中定义的一个查找方法

**4** 实体bean的本地home接口中可以定义多少个创建方法？ （规范：115）

- ❑ A. 0
- ❑ B. 只有1个 — 不一定允许客户创建新实体
- ❑ C. 0或1个
- ✔ D. 0或多个
- ❑ E. 1或多个

**5** 客户可以通过什么办法得到一个现有实体对象远程组件接口的一个引用？（选出所有正确的答案。） （规范：119）

- ❑ A. 调用 `ejbCreate()` — 我们要的是现有的实体
- ✔ B. 由句柄得到引用
- ❑ C. 根据实体对象的主键调用一个方法
- ✔ D. 作为一个方法调用的参数接收引用
- ✔ E. 使用在远程home接口中定义的一个查找方法

**6** 可以对主键类使用什么方法来确定两个键指向同一个实体？（选出所有正确的答案。） （规范：120～121）

- ❑ A. 使用 == 操作符
- ✔ B. 使用`equals()`方法 — 如果两个键通过了equals()测试，它们就是相同的键！而这就意味着相同的实体
- ❑ C. 使用 `isIdentical()` 方法 — isIdentical()用于比较组件接口引用
- ❑ D. 以上都不行

**7** 下面哪些方法可以确定两个实体EJB对象引用指向同样的实体对象？（选出所有正确的答案。） （规范：120～121）

- ❑ A. 使用 == 操作符
- ❑ B. 使用`equals()`方法 — equals()用于比较实体的主键
- ✔ C. 使用`isIdentical()`方法
- ❑ D. 以上都不行

(规范：108)

**8** 关于实体bean远程组件接口的客户视图，下面哪些说法是正确的？（选出所有正确的答案。）

- ☑ A. 多个客户可以并发地访问同一个实体对象
- ❑ B. 可以使用这个接口中的一个方法创建新的实体bean
- ❑ C. 容器崩溃后，实体bean可能不能幸存
- ❑ D. 业务方法不能返回实体对象的一个引用

(规范：116)

**9** 实体bean本地home接口中可以声明多少个查找方法？

- ❑ A. 0
- ❑ B. 只有1个
- ❑ C. 0或1个
- ❑ D. 0或多个
- ☑ E. 1或多个 — 必须有findByPrimaryKey()

(规范：115)

**10** 对于一个本地home接口的 create()方法，以下哪个声明是合法的？（选出所有正确的答案。）

- ❑ A. `public Cust create(int x);` — 需要CreateException
- ❑ B. `public void create(int x) throws CreateException;` — 不能返回void，必须返回组件接口
- ☑ C. `public Cust create(int x) throws CreateException;`
- ❑ D. `public Cust create(int x) throws CreateException, RemoteException;` — 本地接口不能抛出RemoteException

(规范：114)

**11** 对于一个实体bean home业务方法，以下哪个方法名是合法的？（选出所有正确的答案。）

- ❑ A. create
- ❑ B. createCust
- ❑ C. removeAll
- ❑ D. findCust
- ☑ E. selectCust

"find" "create" & "remove" 是保留前缀

# 6 bean/实体同步

## 当个实体bean




**实体bean就是演员。** 只要还活着，实体bean就要么待在池子里，要么扮演着某个人，这是底层持久存储库里的一个人。换句话说，也就是数据库里的一个实体。当bean扮演某个实体时，这个bean必须与底层实体保持同步。如果bean假装是某个人，比方说是Audrey Leone，要是有人在数据库里降低了Audrey的信贷限额……但是忘记告诉bean了。这个bean扮演着Audrey，会大大方方地超额购买，他不知道已经超出了Audrey的当前限额。或者，如果一个客户使用这个bean来修改Audrey的地址，但是bean只是自己留有这个新信息，却没有告诉数据库……唉呀，这可不太妙！

# 实体bean生命周期

## 标准大纲：

6.1 有关bean提供者对于CMP的视图和编程合约，明确哪些说法或例子是正确的，哪些不正确，包括CMP实体bean的需求。

6.6 明确CMP实体bean必须实现哪些接口和方法，以及不应实现哪些接口和方法。

7.1 有关CMP实体bean的生命周期，明确哪些说法或例子是正确的，哪些不正确。

7.2 给定一个方法列表，关于CMP实体bean，明确bean提供者的目的、行为和职责，这些方法包括（但不限于此）：setEntityContext()、unsetEntityContext()、ejbCreate()、ejbPostCreate()、ejbActivate()、ejbPassivate()、ejbRemove()、ejbLoad()、ejbStore()、ejbFind()、ejbHome()和ejbSelect()。

7.3 给定一个方法列表，关于CMP实体bean，明确容器的目的、行为和职责，这些方法包括（但不限于此）：setEntityContext()、unsetEntityContext()、ejbCreate()、ejbPostCreate()、ejbActivate()、ejbPassivate()、ejbRemove()、ejbLoad()、ejbStore()、ejbFind()、ejbHome()和ejbSelect()。

## 具体含义：

只要是与实体bean有关，都会涉及这个要求，所以必须全面地掌握，包括CMP实体bean生命周期的细节，javax.ejb.EntityBean的容器回调方法，bean类中必须写什么，以及bean能从其EJBContext得到什么。

必须知道实体bean中的钝化与会话bean钝化有所不同，因为实体bean会在ejbPassivate()之后返回一个池，不过，回到池里的bean还是活动的对象。你还要知道，实体bean创建与会话bean创建也是完全不同的,在一个实体bean上调用create()意思是“向底层持久存储库插入一个新实体”。另外要知道对实体bean调用remove()相对于会话bean调用remove()要惨烈得多，实体bean删除方法会从数据库中删除实体！

你要会看实体bean方法，知道在哪种情况下调用这个方法，以及在这个方法中应该做些什么。例如，需要知道，在bean的ejbCreate()方法中，这个bean不能得到其EJB对象的一个引用，不过在ejbPostCreate()中是可以的。

你要知道需要在bean中编写哪些方法，而哪些方法要留给容器生成。例如，要知道bean提供者必须为持久字段编写抽象获取方法和设置方法，不过必须只在home接口中定义查找方法，而不要在bean类中定义。

# 实体bean生命周期

| 标准大纲： | | 具体含义： |
|---|---|---|
| 8.1 | 给出一组行为，将其中的行为与负责完成该行为的相应EntityContext方法相匹配。 | 必须了解EntityContext的方法，还要知道在什么情况下可以调用这些方法。例如，应该知道，如果你在运行setEntityContext()方法，就可以使用特殊的JNDI命名空间来查找一个DataSource的引用，但是不允许从这个方法访问底层数据库（这里还不存在有意义的事务，所以容器不允许你访问底层数据库）。<br><br>你应当知道不能在bean上下文上调用getUserTransaction()方法，因为实体bean必须使用容器托管事务，而getUserTransaction仅适用于bean托管事务。 |
| 8.2 | 关于实体bean的主键和对象身份，区别哪些说法是正确的，哪些不正确。 | 你要知道实体bean必须有一个惟一的身份（两个bean不可能有相同的主键），而且CMPbean的主键必须是bean的某个持久字段，或者是一个定制主键类，这个主键类的字段都是bean中定义的容器托管字段。你得知道，这个主键类类型必须是Serializable，而且必须合法地覆盖equals()和hashcode()方法。 |

# 实体bean的真正强大之处在于同步

容器总知道bean和数据库（底层实体）什么时候必须同步，这样不论是谁都不会有“过时”的数据。

实体bean和数据库里的相应行必须保持同步！要记住，bean不是实体，bean只是真实实体的一个表示。bean是一种处理数据的面向对象方法，而数据库中的实体才是真正的实体。如果数据库中的实体死了（也就是说，被删除了），那么对应这个特定实体的bean也会死掉（不过，bean实例还活着，这在上一章已经介绍过）。

容器的任务是确保实体和bean保持同步，以保证没有人会看到一个过时的bean，或者数据库里有过时的行。要记住，实体bean可能不是访问数据库的惟一途径！实际上，大多数情况下，bean提供者（你）在开发的时候没有办法确定你的bean是不是惟一的一条路（别人只能通过你的bean来访问实体数据）。

例如，你的Customer数据库可能被你公司的大量应用所使用，其中一些直接处理数据库，还有一些会通过一个EJB应用中的Customer bean来访问数据库。

如果一个客户有bean的一个引用，这个客户可能会修改bean的状态，比如说通过调用一个设置方法来修改。如果这个设置方法对应于bean的某个持久字段（换句话说，就是表中的一列，如地址），此时bean和数据库会暂时不同步。在bean更新数据库之前，数据库里的不是当前地址！

下面来考虑一下，如果bean缓存了实体的新数据，但是没有告诉数据库，会有怎样的灾难。如果有人运行了另一个应用，请求从数据库得到这个Customer的地址。那就太糟糕了。

如果情况反过来，也同样很不好：倘若有人更新了数据库，bean必须知道！否则，EJB应用中的bean想表示数据库中的实体，但是实际上这个bean并不是实体状态的真实反映。也就是说，bean过时了。这说明，这个bean可能毫无用处。






## 哦，不！
## 实体bean和它表示的实体居然有不同的数据！




有人修改了bean状态，但是没有修改数据库！

| First | Last | Key | Address | Phone |
|---|---|---|---|---|
| Chris | Martin | 100 | **42 Foo st.** | 555-1212 |
| Fran | Healy | 16 | 200 Bar rd. | 555-3434 |
| Bela | Fleck | 5 | 25 Pick lane | 555-5656 |

**对于实体bean，容器最重要的任务就是确保这种情况（bean和数据库不同步）不会导致任何破坏。**

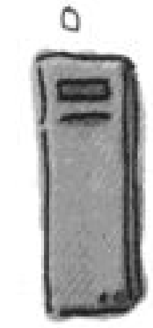
**容器必须确保：**

- 有人处理bean（而且可能修改其状态）时，任何人不得处理数据库中的真实实体。
- 一旦一个实体bean的状态有所更新，在任何人访问数据库中的这条记录之前，数据库也必须更新。
- bean代表一个特定实体运行任何业务方法之前，必须用该实体的状态刷新这个bean。换句话说，对应Joe Bloggs #88的实体bean运行getCreditLimit()方法之前，容器必须用对应Joe Bloggs的最新数据来加载实体bean。否则，这个bean可能返回错误的信贷限额——这只是上一次用数据库中Joe的数据加载bean时所用的值（而不是最新的数据）。

哎唷，你现在让我更担心了。这是不是表示，在每个业务方法调用之间bean都有可能过时？你是说，每次客户在bean上调用一个方法时，你都必须先到数据库去，用数据库里的实际数据来刷新bean的数据，以防数据有修改，是这样吗？如果真地是这样，那你还不如直接去访问数据库呢！



不对！我说的不是这个意思，不过看得出来，表面上好像是这样。但是，这里少了一部分，这就是事务！只要bean的方法作为一个事务的一部分得到调用，这个bean就不必用数据库里的数据来刷新自己的状态。



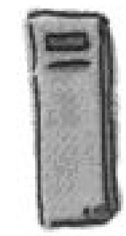
看吧，客户调用一个业务方法，而且这个方法开始了一个事务时，我就会告诉数据库把这个实体锁住！（目前，你可以把它想成是锁住一行，但实际上可能是另外一种稍有不同的东西。）如果实际的实体被锁住了，bean就不会变得过时，因为没有人能通过数据库来访问这个实体。在这个事务结束之前，我不会告诉数据库打开锁的，所以bean很安全。



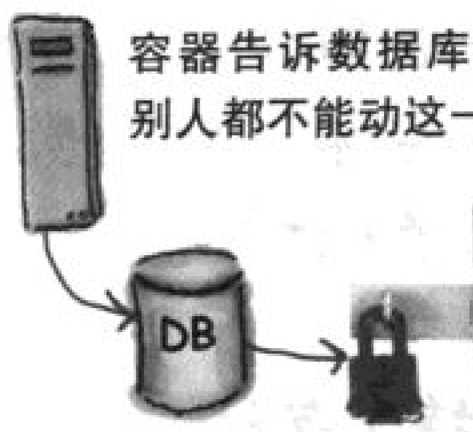
## 实体bean与底层实体如何保持同步

① 

客户调用一个业务方法。

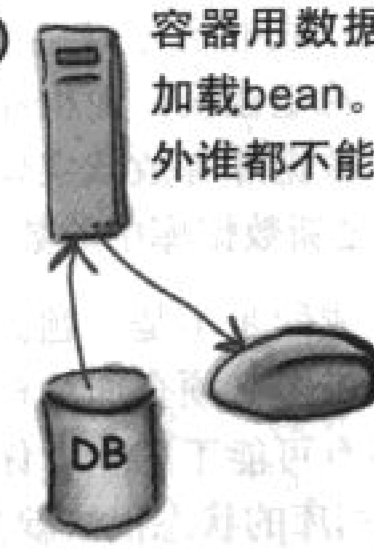
② 

容器截获这个调用，并在得到bean之前先开始一个事务。

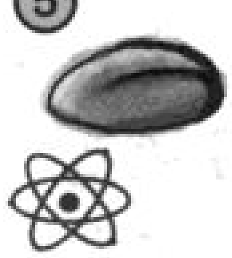
③ 

容器告诉数据库把这一行锁住（除了容器外，别人都不能动这一行）。

| Last | First | PKey |
|---|---|---|
| Poly | Morphism | 72 |
| Dewey | Cheatem | 900 |

④ 

容器用数据库中的实体状态加载bean。现在除了bean以外谁都不能修改实体数据。

⑤ 

bean在同一个事务中运行多个业务方法，它知道数据库中的底层实体数据不会改变（因为对应这个实体的行被锁住了）。

⑥ 

容器结束事务，不过首先它会用bean代表这个实体所缓存的新状态来更新数据库（比如，如果有人在bean上调用了setAddress()）。

⑦ 

容器告诉数据库释放在实体行上加的锁。

| Last | First | PKey |
|---|---|---|
| Poly | Morphism | 72 |
| Dewey | Cheatem | 900 |

# 惟一的问题是：该同步的时候谁来做这个工作

容器总是知道bean和数据库实体什么时候必须同步。这是根据事务来确定的。如果一个客户在bean上调用了一个方法，而且这个方法为bean开始了一个事务，容器会知道，自最后一次利用底层实体的数据加载这个bean以来，数据库中的底层实体可能有所改变。所以容器会要求载入实体的数据，让bean刷新它的状态。

当然，反过来也一样。如果bean完成了一个方法，而且这个方法结束了事务，容器就会告诉数据库，让它释放对数据库中这个实体所加的锁。不过……在这一行被锁住期间，这个bean可能自由地缓存了新的数据，这些新数据表示的是实体新近修改的状态（如一个新地址，或电话号码）。换句话说，客户可能在bean上调用了方法，希望在实际数据库中完成一个更新。所以，容器知道，在告诉数据库释放该实体的锁之前，它必须要求bean用这个新状态来更新数据库中的数据。

不过，至于什么时候完成同步不是问题。之所以说它重要，只是因为，你作为bean提供者必须全面分析业务逻辑，准确地了解什么时候bean和实体有可能不同步。你可以想像一下，如果bean和实体不同步，数据库的状态很快就会被破坏……

真正的问题在于是谁在做数据库访问？这只有两种可能：要么是你，要么是容器。

如果你写了数据库访问代码，就要在容器调用的回调方法中编写JDBC语句（容器告诉你说：“嘿，现在该去数据库了！”这个时候就会调用这些回调方法）。另一方面，如果容器来写数据库代码，就能省去你编写代码的很多时间和精力，而且几乎总能得到更好的性能。

**容器托管持久存储（Container-Managed Persistence，CMP）是指容器负责所有数据库访问代码来完成同步，包括增加和删除实体（就是数据库中的记录/行）。**

**bean托管持久存储（Bean-Managed Persistence，BMP）是指由你来写数据库访问代码（JDBC语句），这样在容器告诉你该访问数据库的时候可以使用这些代码。**

**在EJB 2.0中，应当使用CMP。这会免去你很多工作，而且几乎总能提供更好的性能。**

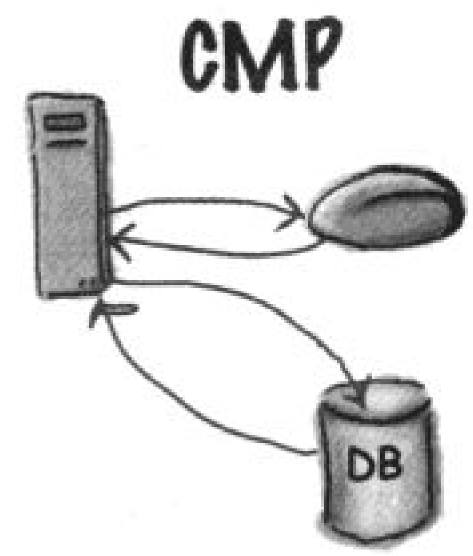
# 容器托管持久存储与bean托管持久存储

## 容器托管持久存储(CMP)

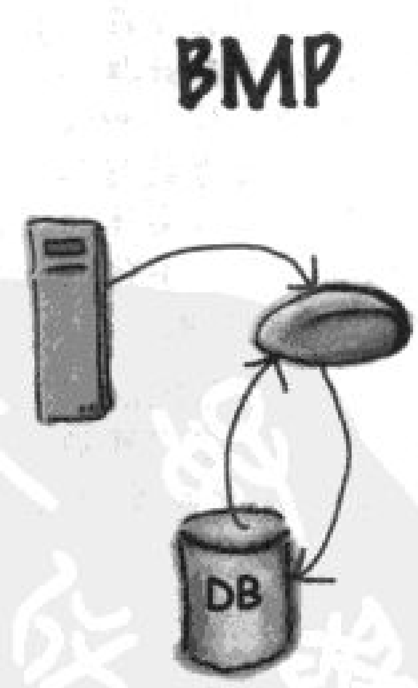
- 在EJB 1.1里还很不完善；但在EJB 2.0中得到很大改进。
- bean提供者设计实体bean类，选择bean的哪些字段是bean持久状态的一部分。持久字段与一个或多个数据库表中的列映射。
- 容器跟踪对bean状态的修改，并根据需要更新真实的实体。例如，如果客户在bean上调用setLastName()，容器就会知道数据库中的实际实体必须更新，这样如果别人访问数据库要得到这个实体（包括使用其他方法访问数据库的非EJB客户），它们就会看到实体的当前状态。容器要根据事务的状态来决定如何保持bean和实际实体的同步（后面会介绍更多有关内容）。
- bean提供者编写EJB-QL来告诉容器如何完成选择。EJB-QL有点像SQL的一个子集，不过具备了一些面向对象特性。EJB-QL可以帮助在bean的OO世界和实际实体的关系世界之间搭建一座桥梁。
- 使用部署描述文件中的信息，容器会编写CMP bean的具体实现，包括实现查找方法和选择方法，以及所有数据库访问代码。换句话说，如果你是一个编写CMP bean的bean提供者，你就不用自己查找DataSource的一个资源连接工厂，不用得到一个SQL连接，也不用编写任何JDBC代码。不仅数据访问代码都由容器负责，而且可以相信，容器很清楚什么时候访问数据库。

## bean托管持久存储（BMP）

- bean提供者编写数据库访问代码，包括查找DataSource（数据源）、得到一个连接，并向数据库发送JDBC语句。这比根本不使用实体bean还是要好，因为容器至少会告诉BMP bean什么时候访问数据库，所以，对于BMP，不用在其中放入逻辑来保持bean和数据库同步。容器告诉你该更新数据库的时候，按它说的做就行了。




容器需要插入一个新实体、删除一个实体、用新的实体状态更新数据库（也就是说，通过bean，实体行中的一列或多列有所改变）的时候，就会访问数据库。




bean需要对数据库做某个工作的时候，容器会在这个bean上调用一个容器回调方法，而由bean完成具体的JDBC工作。

## there are no Dumb Questions

**问：** **容器不是还要给你数据库连接吗？这不正是使用资源工厂引用查找一个DataSource的目的所在吗？如果是这样，那又怎么能完成bean托管持久存储呢？ 或者说，BMP如果是回避这些问题的办法，是不是同时也失去了容器提供的连接池等等服务……**

**答：** 别着急，慢慢来。利用BMP，你仍然可以从容器得到数据库连接，正如你所说，可以通过使用一个资源工厂引用查找javax.sql.DataSource来做到（你必须已经读过有关EJB环境的一章）。你是对的，确实需要从容器管理的一个连接池得到一个连接。但是，一旦得到连接，对于一个BMP bean，下面就要由你自己向它提供语句来完成插入、删除、更新和选择操作了。

另外，尽管我们在谈实体bean，但是要记住，数据库访问并不只针对实体bean。任何bean都可以把数据库访问作为其业务逻辑的一部分。会话bean和消息驱动bean很有可能会在数据库中查找某个东西，或者要在数据库中存储些什么。实体bean和另外两种bean之间存在着差别，这只是因为数据库中存放的是某种东西，这正是（实体）bean所表示的。所以根据定义，实体bean与持久存储库中的实体是绑定的。

消息驱动bean和会话bean不表示数据库中的东西，不过，它们可能需要使用数据库中的实体作为其业务逻辑的一部分。

**问：** **看上去，利用BMP性能好像能好些，至少不会差。BMP是不是能让你有更多的提供？**

**答：** 利用BMP，确实灵活性更大一些，但是有更多的控制则不一定。根据对主流J2EE服务器的几乎所有基准测评来看，BMP的性能并不见好。听上去可能有点自相矛盾，但是，要记住，容器能做的一些事情，你在bean代码中可能做不到。比如说把多个数据库调用绑在一起作为一个调用，使用本地代码更快地完成数据库读写、懒加载、脏检测（当然，规范对这些都未做保证），还有许许多多其他的技巧。

我们知道，这与我们以往的常识有违背，以前我们总说，如果你想做把事情做好，就要自力更生。但这一次有所不同，要记住，服务器开发商都在努力提高自己的竞争力，想要把你的企业划到自己的“势力范围”中。他们比你更清楚性能问题。2.0规范增加而且修改了许多CMP规范，根本目的就是让开发商有更大的空间来实现CMP的优化。




# CMP 2.0演进简史

从EJB规范1.1版以来，容器托管持久存储（CMP）和bean托管持久存储（BMP）就都可以使用。从最高层次上看，CMP和BMP的差别就在于是谁编写数据库访问代码。

对于EJB 2.0，这二者仍然都可以使用，但是现在一般没什么理由再使用BMP了。原来的EJB 1.1规范中有关CMP实体bean的部分很不完善，很麻烦，效率低下，而且功能有限。这些都不是太好。另外，尽管许多开发商可以克服CMP 1.1的一些问题，但是这些解决方案都不属于规范，所以作为bean开发人员，你就有两种选择，要么使用标准CMP来保持可移植性，要么超出规范（可移植性会降低）来得到更好的性能和更优的特性。

建立EJB 2.0规范时，规范小组花了很多时间和精力与最终用户顾客（bean开发人员）和容器开发商交流，无论是bean开发人员和容器开发商都在喋喋不休地说着CMP存在的问题。他们都相当详细地向Sun提出了建议："我跟你说，应该对CMP bean规范做如此这般的修改……"

EJB规范小组听取了这些意见，然后在建议基础上设计规范，再听取意见，再设计规范。最终，他们终于拿出了一个成果，但是这个成果相当糟糕，谁都不喜欢。开发商觉得不好，在开发人员那里也不讨好。而我们这些人由于必须向顾客解释这项新技术，更是反感它。所以随着最后期限的临近，规范小组推翻了他们原先对CMP所做的大部分设计，提出了一个简洁得多的方案。

在这个新的EJB 2.0规范中，CMP比EJB 1.1中要轻便很多（当然也比那个没发行的EJB 2.0规范第一版更简洁）。如今，大多数使用实体bean的开发人员都会使用CMP。实际上，我们的考试根本不涉及BMP，现在之所以还有BMP，主要是出于对既有系统的支持，或者是为了支持一些非常特殊的情况。

由于CMP的巨大转变（原因将在后面分析），所以在本书中不会介绍BMP。从现在开始，我们就假设只讨论CMP。对于CMP和BMP来说，其生命周期有很大不同，而且开发人员的相应职责也截然不同，所以不要忘了，即使我们没有明确地说所指的是CMP，但这里都是从CMP bean的角度来讨论的。




**考试中不会考EJB规范的历史，也不涉及大家对EJB规范的微辞。**

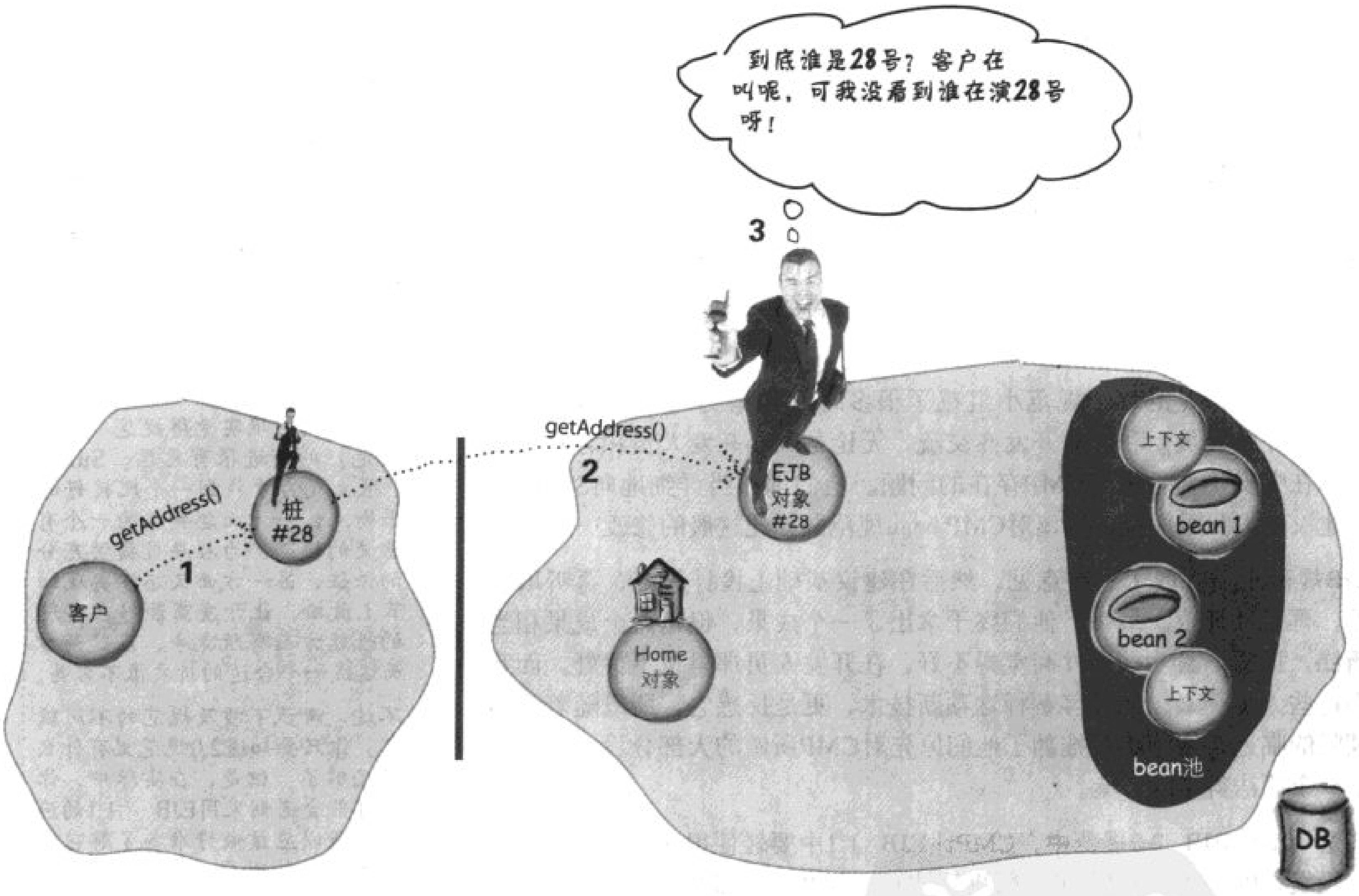
虽然说点这方面的内容会让这本书更有意思……但是，我们还有正事要说呢。

不过，我们确实觉得规范（任何规范）的演进很有意思。Sun花了很大气力才找到一个比较好的平衡点，一方面想要成为一个有意义的规范，为程序员提供充分的保证，另一方面又想在实现细节上放松，让开发商能够在实现的性能方面有所竞争，在这两方面达成一个合适的折衷很不容易。

不过，考试不涉及规范的不同版本。你只要知道2.0规范里有什么就足够了。但是，在实际中，你很可能会遇到采用EJB 1.1的应用，所以应该做好准备了解它们的不同。特别是，不知道算是幸运还是不幸，要是让你负责把应用移植到一个2.0兼容的服务器上，就更得做好准备。;)

## bean演员怎么成为一个bean实体

场景：客户想得到一个特定CMP Customer实体（主键为#28）的地址。



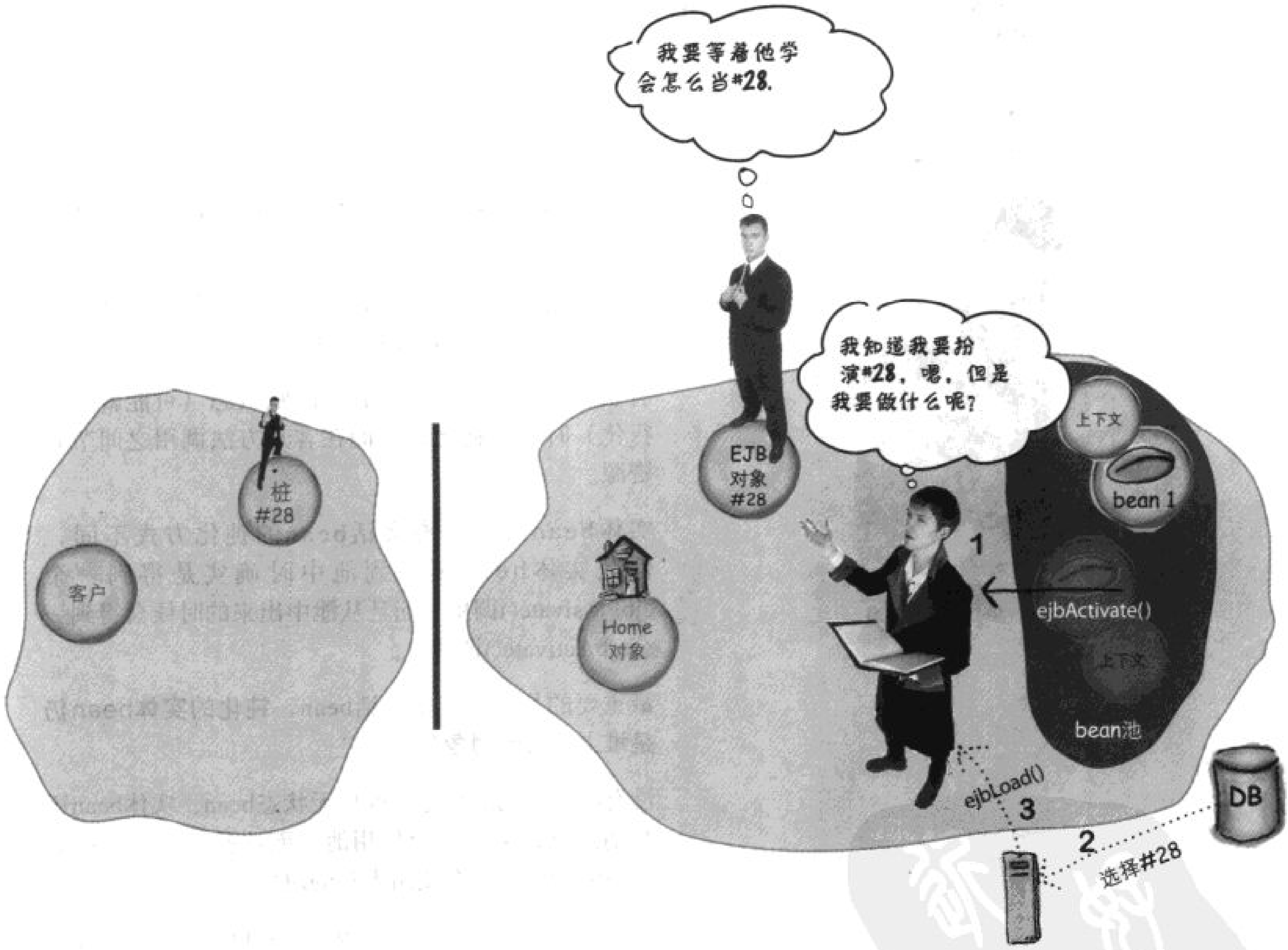

**1** 客户在实体bean #28的桩上调用getAddress()。这个桩是客户在home桩上的一个调用返回的（稍后就会说明客户怎么先得到这个桩）。

**2** 这个调用传递给EJB对象。

**3** EJB对象得到调用，而且着急起来，因为并没有对应#28的实体bean！EJB对象是#28的代理/保镖，但是他没有与一个扮演#28的bean关联。

## bean演员怎么成为一个bean实体

CMP bean从池中出来，准备扮演#28。




**1** 容器从池中拉出来一个bean，然后调用ejbActivate()，将其激活。

**2** 容器在数据库上针对主键#28完成选择，得到实际的实体数据，放到这个bean中（这些数据将成为bean的状态）。

**3** 容器用数据库中的实体数据加载bean，然后调用ejbLoad()来告诉bean："嘿，bean，你已经被加载好了。"




## 相对于会话bean，实体bean的钝化和激活含义完全不同！

无状态会话bean无需钝化就可以回到池中（换句话说，不用得到一个ejbPassivate()调用）。

有状态会话bean被容器置为睡眠状态（可能就是串行化）时会被钝化，从而在客户方法调用之间节省资源。

实体bean与有状态会话bean的钝化方式不同，但是实体bean要回到池中时确实是得到一个ejbPassivate()调用（而且从池中出来的时候会得到一个ejbActivate()调用）。

最重要的是，不同于会话bean，**钝化的实体bean仍是堆上的活动对象！**

没有bean在池中睡觉一说！无状态bean、实体bean还有消息驱动bean都会使用池。而这些池中放的都是堆中活动的、正在使用内存的对象。

只是有状态bean会置为睡眠状态（而这被有些含糊地称为钝化），但它与池没有任何关系。

所以，他们（规范小组）确实让“钝化”这个词一词多意，可能就是让你容易混淆，然后需要更多的EJB书籍吧（对于这一点，作为写书的我们倒是感激不尽）。

# ejbPassivate()：会话和实体bean之间的差别

## 实体bean




- 当bean完成了一个业务方法（除了remove()），要回到池中的时候会调用ejbPassivate()。在钝化状态，bean没有身份，它不表示数据库中的任何实体！
- 用于释放一些资源，你不希望bean待在池中没有运行业务方法的时候还浪费这些资源（一般地，这个方法为空）。
- 实体bean不像会话bean那样钝化。换句话说，bean不会串行化或保存，所以不需要把不可串行化对象的引用置为null。
- 需要一个bean为客户的一个业务方法调用提供服务时，容器会调用这个bean的ejbActivate()方法。bean在运行触发激活的业务方法之前要运行ejbActivate()（这个方法通常也是空的）。

## 有状态会话bean



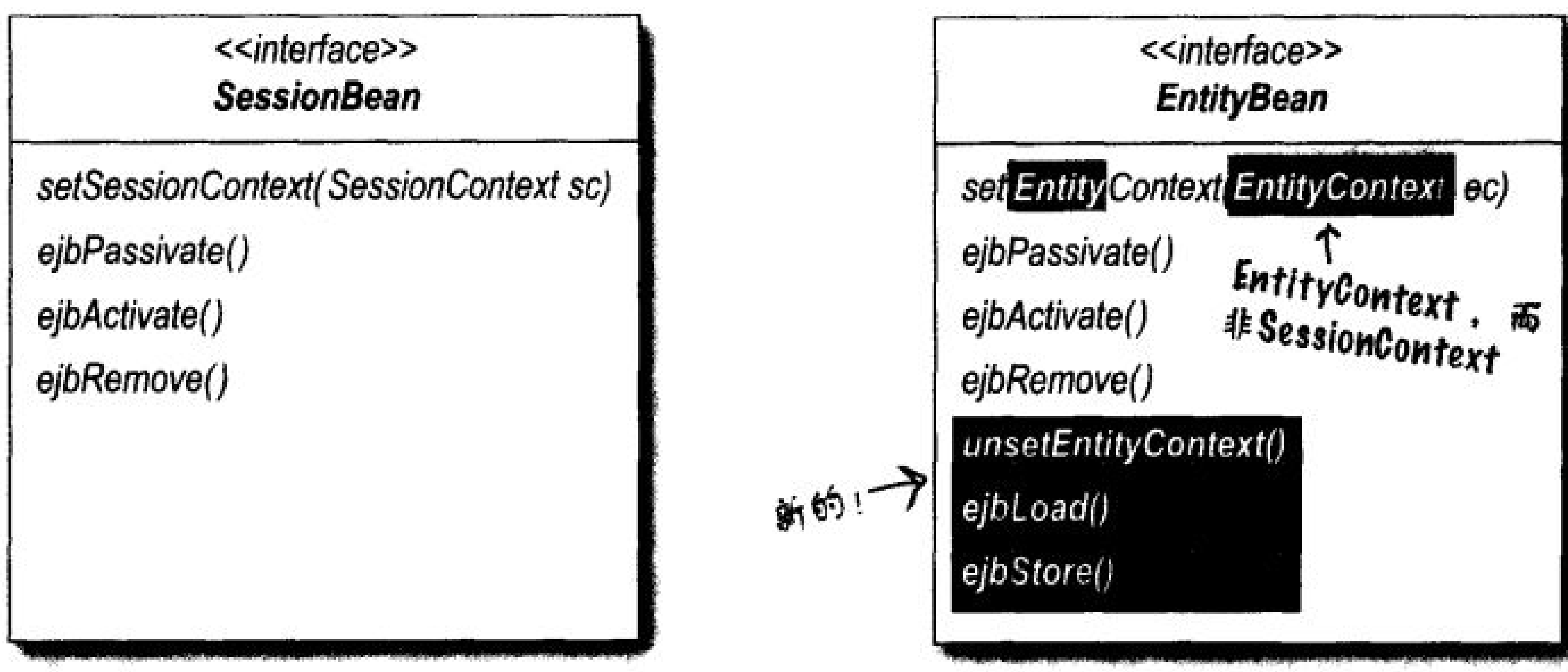

- 容器决定在客户的业务方法调用之间节省资源时会调用ejbPassivate()。
- 用于释放资源，并做好准备，使bean的状态可以串行化（如果是不可串行化的引用，要置为null，等等）。
- 这个bean要"置入睡眠状态"，不再占用堆上的空间。
- 客户在EJB对象上调用一个业务方法时，容器会调用这个bean的ejbActivate()方法。bean在运行触发激活的业务方法之前要运行ejbActivate()。

## 无状态会话bean

不适用……无状态会话从不钝化

- 不适用！无状态bean有一个池，但是激活和钝化与池无关。要记住，无状态会话bean永远也得不到ejbActivate()或ejbPassivate()调用。

# EntityBean接口增加了另外3个新的容器回调（其中两个只是为了同步）




EntityBean接口增加了3个新方法（而且上下文设置方法也有所修改，这里要为bean提供一个EntityContext而不是SessionContext）。但是……（这个但是很重要）……即使是两个接口中同样的方法，它们的表现也不一样。

对于SessionBean中的4个方法，只有上下文设置方法的表现与EntityBean中的上下文设置方法相当。另外3个（ejbPassivate()、ejbActivate()和ejbRemove()）都有截然不同的含义。

*这一点后面就会看到。*

现在先把你原来对激活、钝化和删除的理解扔到一边。与会话bean相比，这些概念对实体bean来说含义是完全不同的。

# 即使方法相同，表现也不一样

## SessionBean接口

***setSessionContext(SessionContext sc)***

容器为bean提供一个上下文引用。

***ejbpassivate()***

应用于有状态会话bean，当bean要被串行化（或串行化之类的钝化）时，会调用这个方法。

***ejbActivate()***

应用于有状态会话bean，当bean在钝化之后要被重新激活（可能是逆串行化）时，会调用这个方法。

***ejbRemove()***

对于有状态会话bean，当客户调用remove()时，会在bean上调用ejbRemove()。对于无状态bean，当容器想缩小池的大小时，会在bean上调用这个方法。

在SessionBean和EntityBean接口都有的4个方法中，只是上下文设置方法的表现相同！对于实体bean，激活、钝化和删除都与会话bean的激活、钝化和删除完全不同。

## EntityBean接口

***setEntityContext(EntityContextec)***

容器为bean提供一个上下文引用。

***ejbPassivate()***

bean在事务结束之后要返回到池中时会调用这个方法。

***ejbActivate()***

从池中取出bean，为客户的业务方法调用提供服务时要调用这个方法。

***ejbRemove()***

客户调用remove()，想从数据库中删除这个实体时调用这个方法！

***unsetEntityContext()***

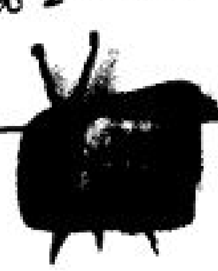
容器想缩小池的大小时调用这个方法。

***ejbLoad()***

bean已经用底层持久存储库的数据刷新时调用这个方法。

***ejbStore()***

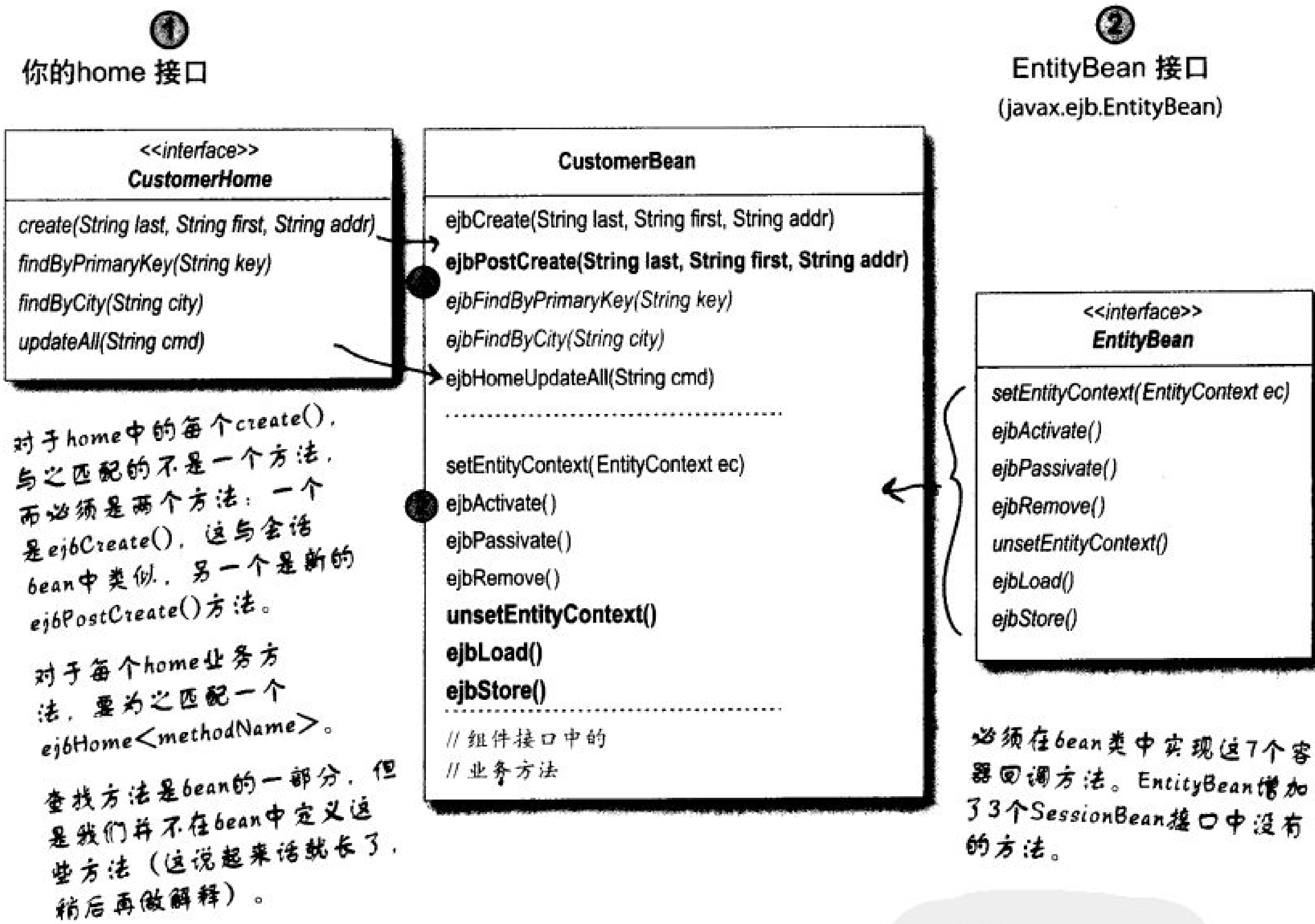
当容器要更新数据库来反映bean状态时调用这个方法。



**EntityContext 增加了 getPrimaryKey()**

SessionContext和EntityContext都扩展了EJBContext。EntityContext中只有一个方法是SessionContext没有的，这就是getPrimaryKey()。这里有点问题，因为EJBObject和EJBLocalObject接口中都有一个getPrimaryKey()，所以会话bean客户和实体bean客户都能看到这个方法（不过，会话bean客户要是调用这个方法就会得到一个异常）。但对于上下文接口，只有实体bean才能看到bean上下文中的getPrimaryKey()。

# 先等等……还不仅如此！实体bean还有新的home容器回调方法




**对于任何使用容器托管持久存储（CMP）的实体bean来说，至少有7个容器回调方法**，这些方法都来自EntityBean接口实现。在home中不必非要有create()方法，但是如果确实有create()方法，对于每个create()，与之匹配的就不是一个方法，而必须是两个方法：一个是ejbCreate()，另一个是ejbPostCreate()。如果有home业务方法，则必须在bean类中编写一个匹配的ejbHome<methodName>方法。

如果使用bean托管持久存储（BMP），也就是你要自己编写数据库访问代码，那么还必须对每个查找方法编写一个匹配的ejbFind<whatever>方法。要记住，home接口惟一必要的方法是findByPrimaryKey()。但是在CMP bean中，即使home接口中有查找方法，也不用在bean类中放这些查找方法。

# 编写CMP实体bean：置其为抽象（abstract）



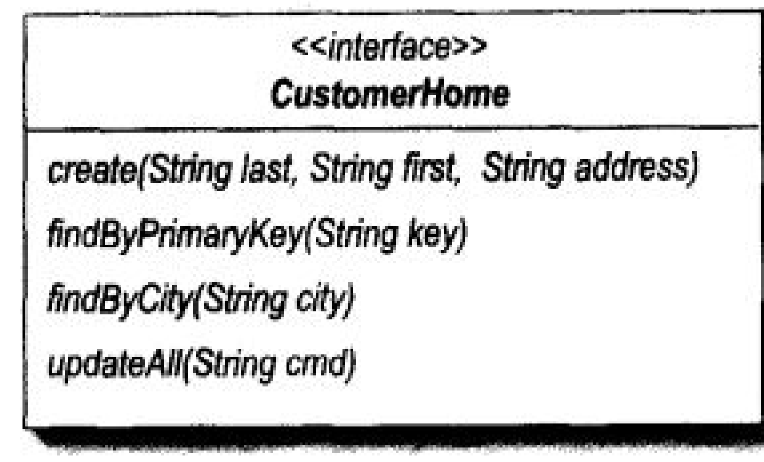

置CMP实体bean类为抽象（abstract）。

还是需要你来实现javax.ejb.EntityBean的容器回调，而且如果你必须自己写所有的业务方法（包括home业务方法），另外如果有创建方法，那你还必须在bean类中编写匹配的方法。但除了这些，还有一大堆的事情等着容器去做，这包括为你的持久字段编写存取方法（获取方法和设置方法）。例如，如果你的 Customer表有一个地址列，你把这个列映射到bean类中的一个字段（举例来说，这样客户就能调用getAddress()），你不用自己编写那个字段的获取方法和设置方法。实际上，你甚至不用声明这个字段！利用CMP，只需定义获取方法和设置方法，就创建了一个虚字段。换句话说，之所以说你的bean中存在对应地址的持久字段，只是因为这里有地址的一个抽象获取方法和一个抽象设置方法（嗯……这不完全对。除此以外，部署描述文件中还有相应的一项，不过这个内容到下一章再介绍）。

# 在bean类里要放3种东西：

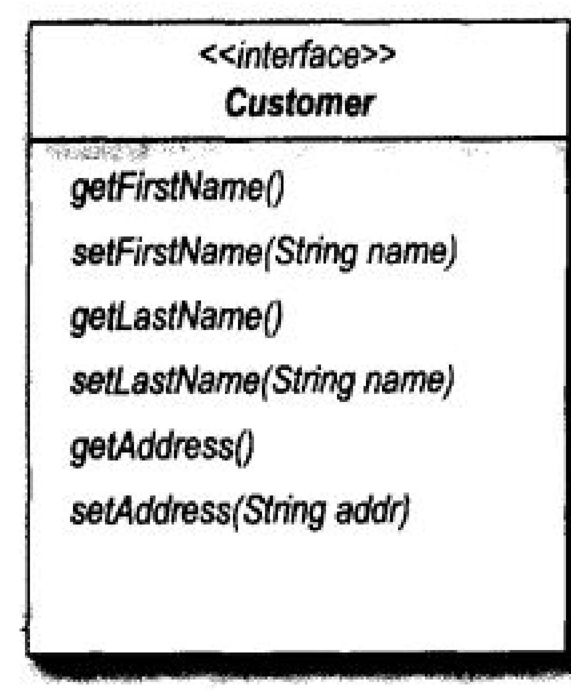
## ● 来自home接口的方法

- 对于home中的每个create()方法，在bean类中必须有一个匹配的ejbCreate()和一个ejbPost-Create()方法。
- 对于home中的每个业务方法，bean类中必须有一个匹配的ejbHome<method>方法。
- 对于CMP bean，不用编写匹配的查找方法。容器会根据你在部署描述文件中放的信息为你编写这些方法（下一章我们会详细介绍这个内容）。



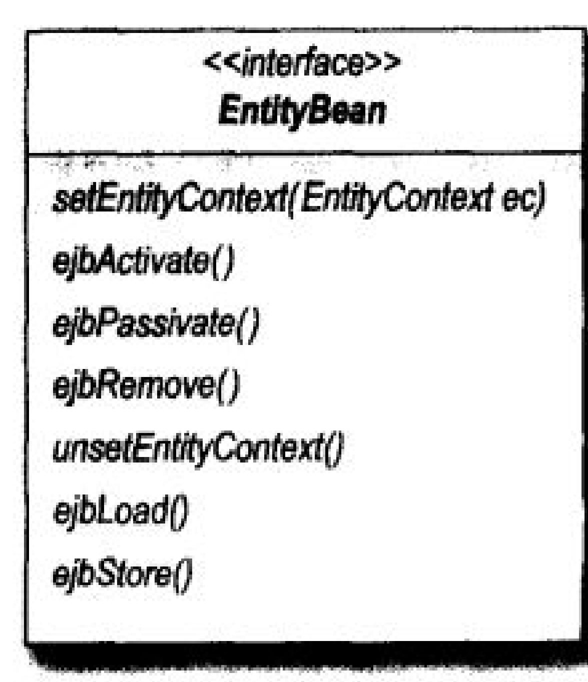

## ● 来自组件接口的方法

- 对于组件接口中的每个方法，在bean类中必须有一个匹配的具体方法实现。




## ● 来自EntityBean接口的方法

- 必须直接或间接地实现EntityBean接口。所以除非有一个实现了所有方法的超类，否则你就要在bean类中负责编写具体的方法实现。



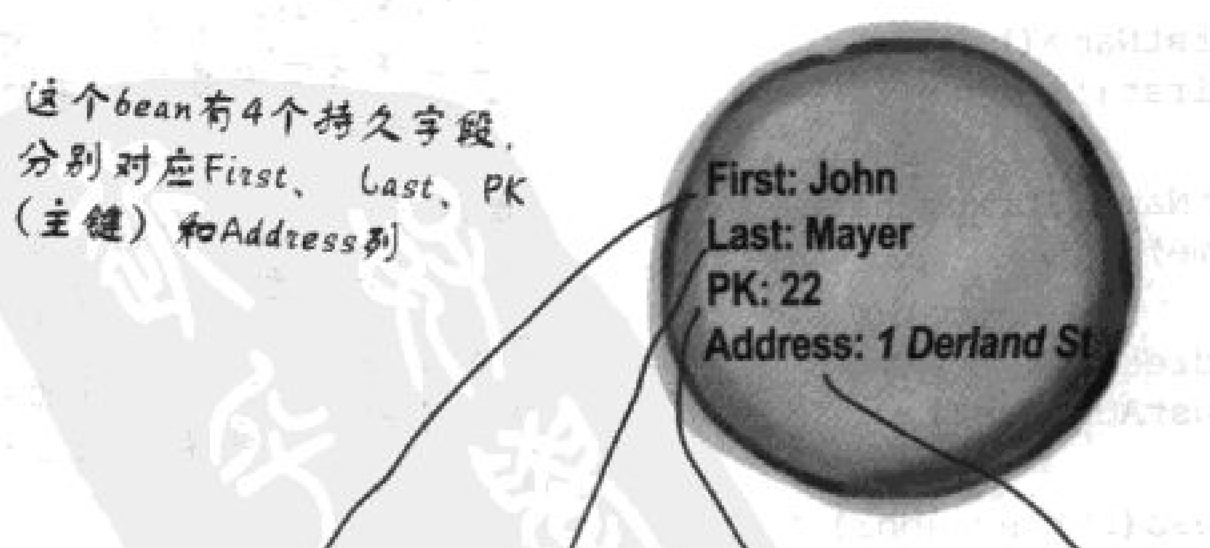
# 另外……(对了，应该是4种东西……)

## ④ 虚持久字段

- 对于每个持久字段，提供一个抽象获取方法和一个抽象设置方法。




**“虚持久字段”是指映射至数据库中某个列的值。它们表示了实体的持久状态。在bean类代码中，这些虚持久字段仅作为抽象获取方法和设置方法存在。**




| First | Last | PK | Address |
|---|---|---|---|
| John | Mayer | 22 | 1 Derland St. |
| Fran | Healy | 16 | 200 Bar St. |
| Bela | Fleck | 5 | 25 Pick Lane |

# 虚字段不是实例变量！

```
public abstract class CustomerBeanCMP implements EntityBean {

   private EntityContext context;

   public String ejbCreate(String last, String first, String addr) {
     this.setLast(last);
     this.setFirst(first);
     this.setPK(makePK());
     this.setAddress(addr);
     return null;
   }

   public abstract String getLast();
   public abstract void setLast(String last);
   public abstract String getFirst();
   public abstract void setFirst(String first);
   public abstract String getCustAddress();
   public abstract void setCustAddress(String addr);
   public abstract String getPK();
   public abstract void setPK(String pk);


   public String getLastName() {
      return this.getLast();
   }
   public void setLastName(String name) {
      this.setLast(name);
   }
   public String getFirstName() {
      return this.getFirst();
   }
   public void setFirstName(String name) {
      this.setFirst(name);
   }
   public String getAddress() {
      return this.getCustAddress();
   }
   public void setAddress(String addr) {
      this.setCustAddress(addr);
   }

   // 来自EntityBean和home接口
   // 的更多方法
}
```

注意：惟一的实例变量是对应EntityContext的变量！

这些是虚持久字段。它们只是一些抽象的获取方法和设置方法，映射至数据库中的列，表示了实体的持久状态。这也是实体要保存的东西！

这些是来自组件接口的方法，要展现给客户。

我们知道你在想什么……为什么又来了一组获取方法和设置方法，为什么这些方法不能在组件接口中展现，而只能展现上面的抽象存取方法？

是这样，这些方法确实可以展现，但这种想法并不好，自己考虑考虑吧……

下面给你一个提示：想像一下，这些非抽象方法中的代码如果比你现在看到的多得多，会怎么样呢？

更明显的提示：想像一下，如果非抽象方法有验证代码的话……

更大的提示：想想封装。

## Sharpen your pencil

使用下面的接口，编写一个合法的bean类。不必编写具体的业务逻辑，但是至少要列出bean类中必须编写哪些方法，而且要提供正确的声明。

<<interface>>
**ProductHome**

*create(String description, String cat, double price, String ID)*
*findByPrimaryKey(String key)*
*findByCategory(String category)*
*getLowStockItems()*

<<interface>>
**Product**

*getCategory()*
*getID()*
*getDescription()*
*setDescription()*
*getPrice()*
*setPrice()*

<<interface>>
**EntityBean**

*setEntityContext(EntityContext ec)*
*ejbActivate()*
*ejbPassivate()*
*ejbRemove()*
*unsetEntityContext()*
*ejbLoad()*
*ejbStore()*

在下面写出这个类；不用写导入语句。

# CustomerBeanCMP类的完整代码

(注意：这里没有加注释，因为这是下一页的练习中要你做的事情)

```
package headfirst;

import javax.ejb.*;

public abstract class CustomerBeanCMP implements EntityBean {

   private EntityContext context;

   public String ejbCreate(String last, String first, String addr) {
     this.setLast(last);
     this.setFirst(first);
     this.setPK(makePK());
     this.setAddress(addr);
     return null;
   }

   public String getLastName() {
      return this.getLast();
   }

   public void setLastName(String name) {
      this.setLast(name);
   }

   public String getFirstName() {
      return this.getFirst();
   }

   public void setFirstName(String name) {
      this.setFirst(name);
   }

   public String getAddress() {
      return this.getCustAddress();
   }

   public void setAddress(String addr) {
      this.setCustAddress(addr);
   }

   public void setEntityContext(EntityContext ctx) {
      context = ctx;
   }
```

```
    public abstract String getLast();
    public abstract void setLast(String last);
    public abstract String getFirst();
    public abstract void setFirst(String first);
    public abstract String getCustAddress();
    public abstract void setCustAddress(String addr);
    public abstract String getPK();
    public abstract void setPK(String pk);

    public void unsetEntityContext() { }
    public void ejbLoad() { }
    public void ejbStore() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() { }

    private String makePK() {
      int rand = (int) (Math.random() * 42);
      return ""+ rand;
    }
}
```

记住：这是世界上最傻的主键生成器。这里
我们假设：就算是返回同样的数也可以接受，
但是这是不可行的，因为主键必须惟一，否则
的话……想想看会怎么样……JVM会毫不留情
地给你一个DuplicateKeyException异常。不过正
如我们所说，在实际中，可以使用客户提供的
一个参数作为主键，也可以使用一种真正的主
键生成引擎，或者要求容器给数据库提供自动
生成的键（如果你的开发商支持这一点）。

## Sharpen your pencil

● 根据方法放在类中的理由，对CustomerBeanCMP类中的每个方法加上标记，也就是以下4个符号之一：

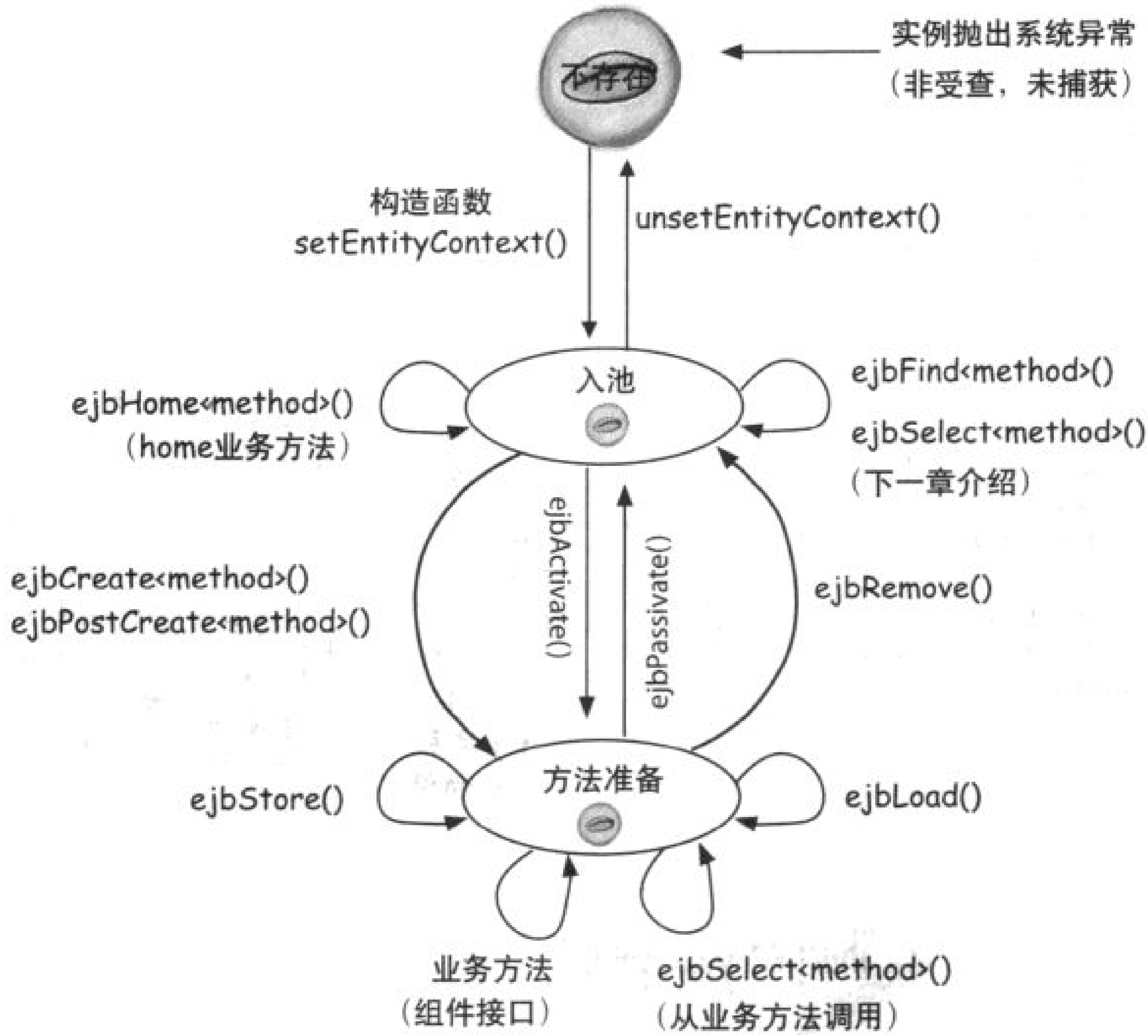
**H C EB VF**

例如，之所以需要ejbCreate()方法是因为home中有一个匹配的create()方法，所以要在ejbCreate()方法旁边加上H。

● 如果编译器会检查某些方法，就要在该方法旁打一个对号。也就是说，如果你没有写某个方法，编译器为此会报错，就对该方法打勾 ✓

● 你还能想到什么细节问题，请自己给这个代码加注释。对于这个练习（不包括前两个），要尽可能地多写点东西，然后再返回去看这一章前面的几页，了解一下哪些少了，哪些要修改。

## 实体bean实例生命周期



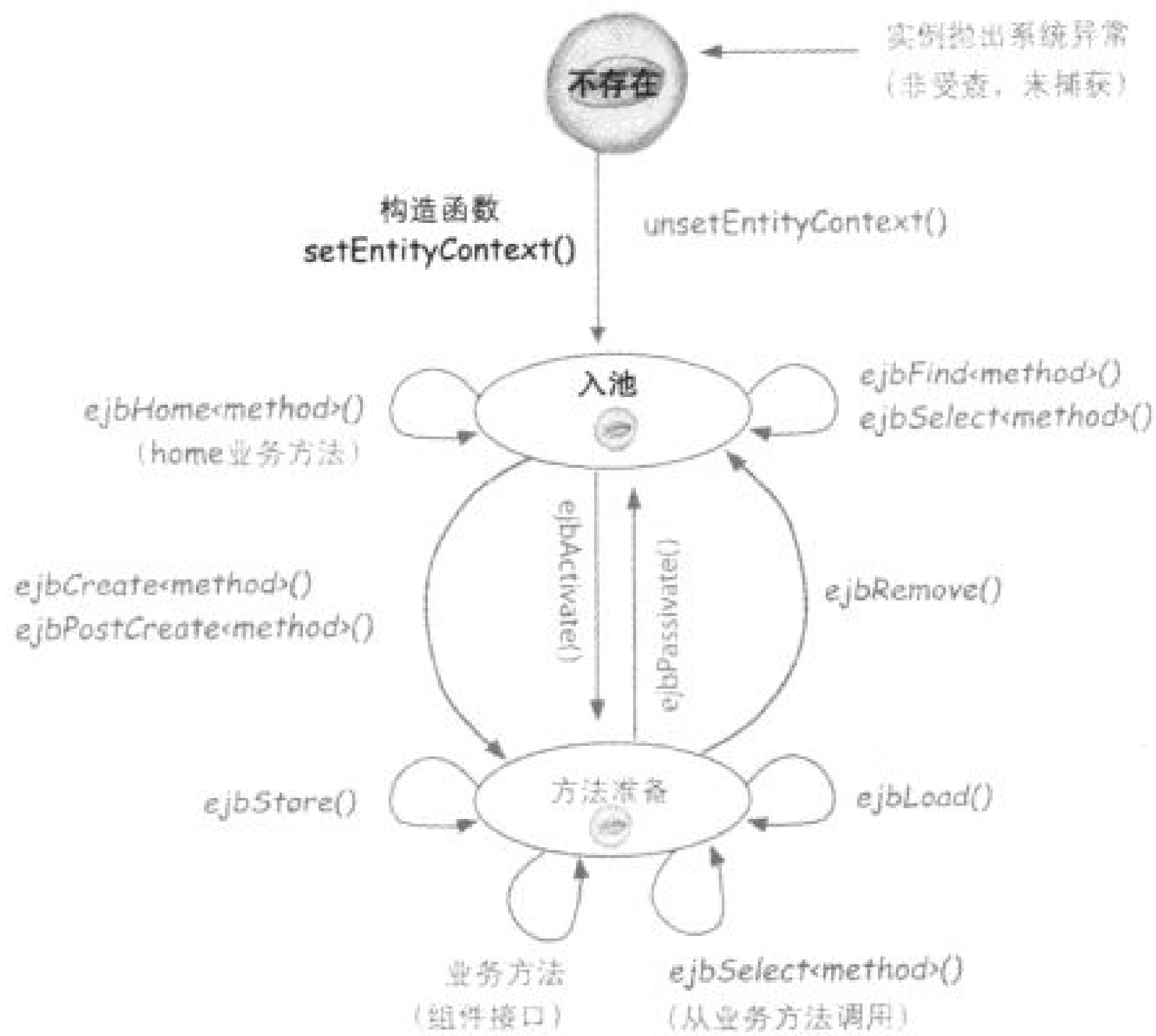




尽管乍看上去情况不太好，但这只是表面现象。放松……等读完这一章，你就能恢复信心，90%都能记住，等学完第七章，你肯定会完全掌握的。

# 实体bean实例迁移

别指望现在就把这个图都记住！这一章大部分都是在介绍这个内容。




**从不存在到入池**

**构造函数**

**setEntityContext()**

**注意：没有create()!**

---

**从入池 到方法准备**

ejbCreate() / ejbPostCreate()

或

ejbActivate()

（容器调用ejbCreate()时不调用ejbActivate()）

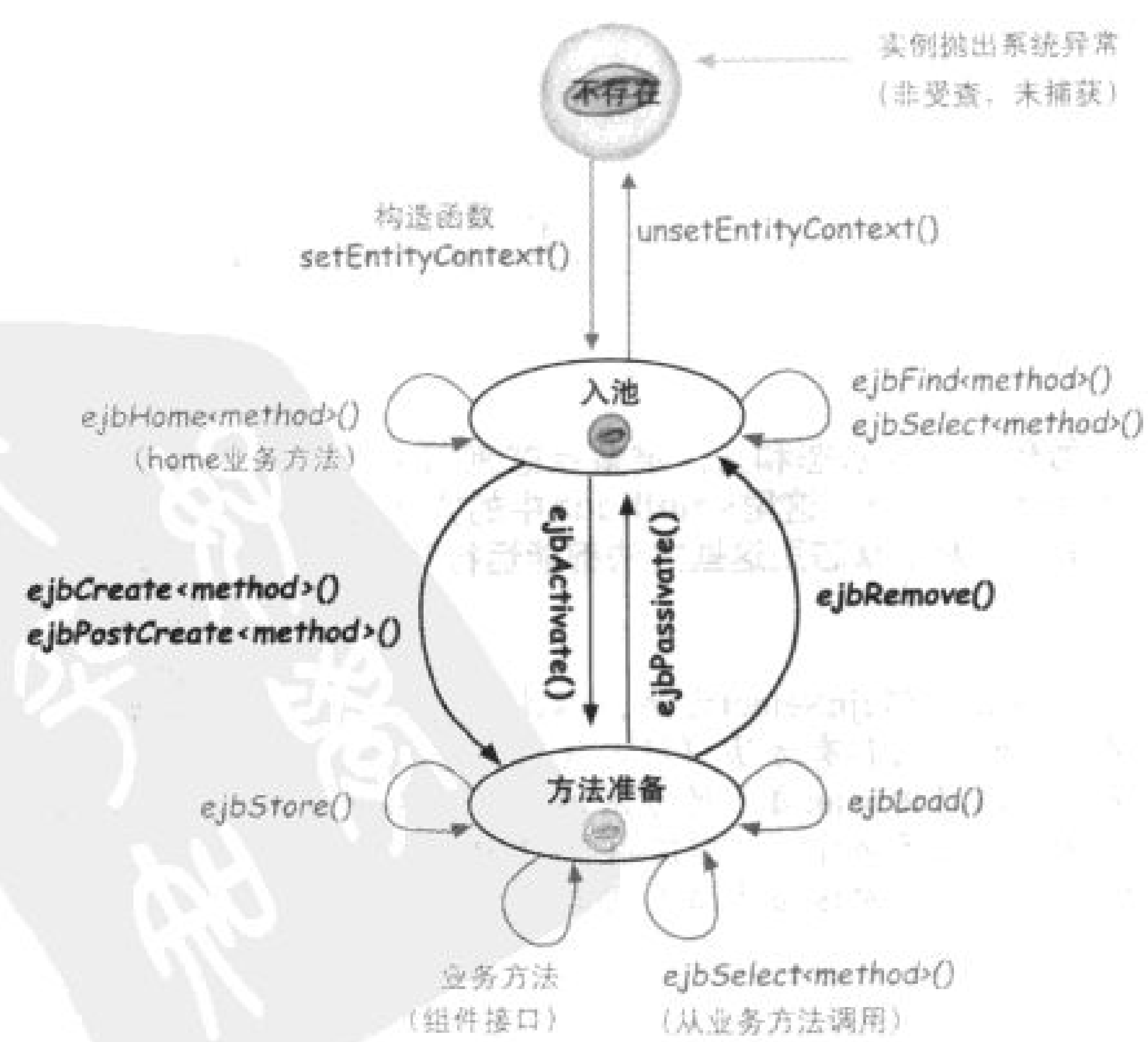
**从方法准备 到 入池**

ejbPassivate()

或

ejbRemove()

（二者只调用其一！调用ejbRemove()的bean在回到池中之前不用钝化！）



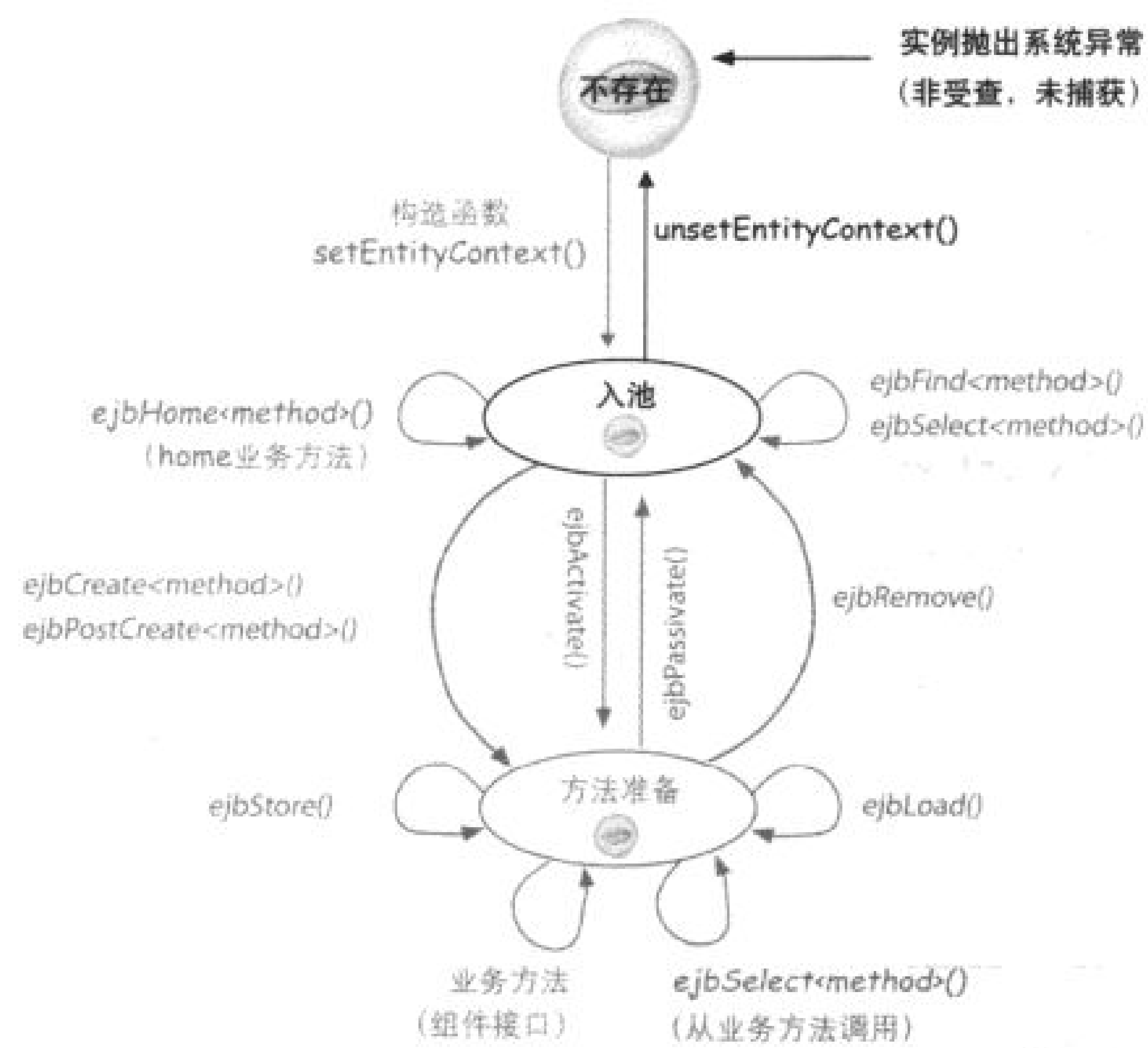
# 实体bean实例迁移

移至不存在

**unsetEntityContext()**

**或**

**实例抛出一个系统异常**




## there are no Dumb Questions

**问：为什么入池状态和方法准备状态都写上了一个ejbSelect<method>（这里<method>中的方法可以任意）？怎么在两个状态里这些方法都能运行呢？**

**答：** 稍后会介绍ejbSelect方法，不过，现在先把它们想成是bean中的私有方法（换句话说，它们不是由客户调用，而只由bean自己的方法调用），这些方法要对数据库做选择操作。它们仅用于CMP bean，可以提供很大方便，因为这些方法是由容器为你实现的。

之所以在两个状态中都能运行这些方法，原因在于这取决于客户在什么接口上调用了某个方法，而这个方法又调用了一个选择方法。所以，如果客户调用了一个 home业务方法，这个home方法又调用了一个ejbSelect<method>，bean就会待在池中运行方法。不过，如果从bean组件接口中的某个方法调用选择方法，这意味着方法是在一个特定的实体上运行的，比如说，在Frank Foof #56实体上运行，所以bean处于方法准备状态。

## there are no Dumb Questions

**问：** **看上去有两种方法可以移到方法准备状态：要么是客户调用一个create()方法，要么是容器调用ejbActivate()。这是不是意味着每次离开池的时候ejbActivate()不一定会得到调用？**

**答：** 完全正确。bean只能通过这两条路（创建或激活）移到方法准备状态，但是绝对不会同时创建和激活。所以，如果你的设计是在ejbActivate()中获得资源，以便bean为一个业务方法服务时资源总是可用的，那你最好把这些资源获取工作放到ejbCreate()里（或者放在ejbPostCreate()中，稍后就会介绍这二者的区别）。

在实际中，EJB 2.0中很少使用ejbActivate()完成太多工作。这一章和最后一章（模式和性能）会更多地谈到这个内容，不过这里可以简单总结一下：可以直接在需要资源的业务方法中获得和释放珍贵的资源，这样往往更加高效。这样一来，你的bean处于活动状态（也就是说，不在池中）但没有真正运行一个方法时，就不会霸占住资源（不让别的bean访问）。不错，相对于把这些工作放在ejbActivate()中执行一次的做法，这意味着每个业务方法中会增加一些开销，但在许多情况下，获取资源的开销可谓微乎其微，可以想想……比如资源是池中的数据库连接，要让数据库连接打开时间过久，而你早已经不需要访问这些资源了，这就会存在很大的可扩展性开销，相比之下，获取资源的开销可算是小巫见大巫了。

关键是：你可能会发现ejbActivate()是空的，如果是这样，就不用担心通过ejbCreate()调用从池中出来的时候漏掉ejbActivate()，也就是没有执行ejbActivate()也没关系，因为它本身并没有做任何工作。

## Sharpen your pencil

考试时，必须知道EntityBean接口中有哪些容器回调方法，所以必须记住。比较难的是，有些方法与SessionBean接口中相应的会话bean方法是同名的，但是表现却大不相同。先别翻到下一页看答案！

1. 客户调用这个方法告诉容器，他（客户）已经用完了这个bean的EJB对象引用：

______________________

2. bean回到池中时，从数据库删除了一个实体之后调用这个方法：

______________________

3. bean构造函数运行之后紧接着就会调用这个方法：

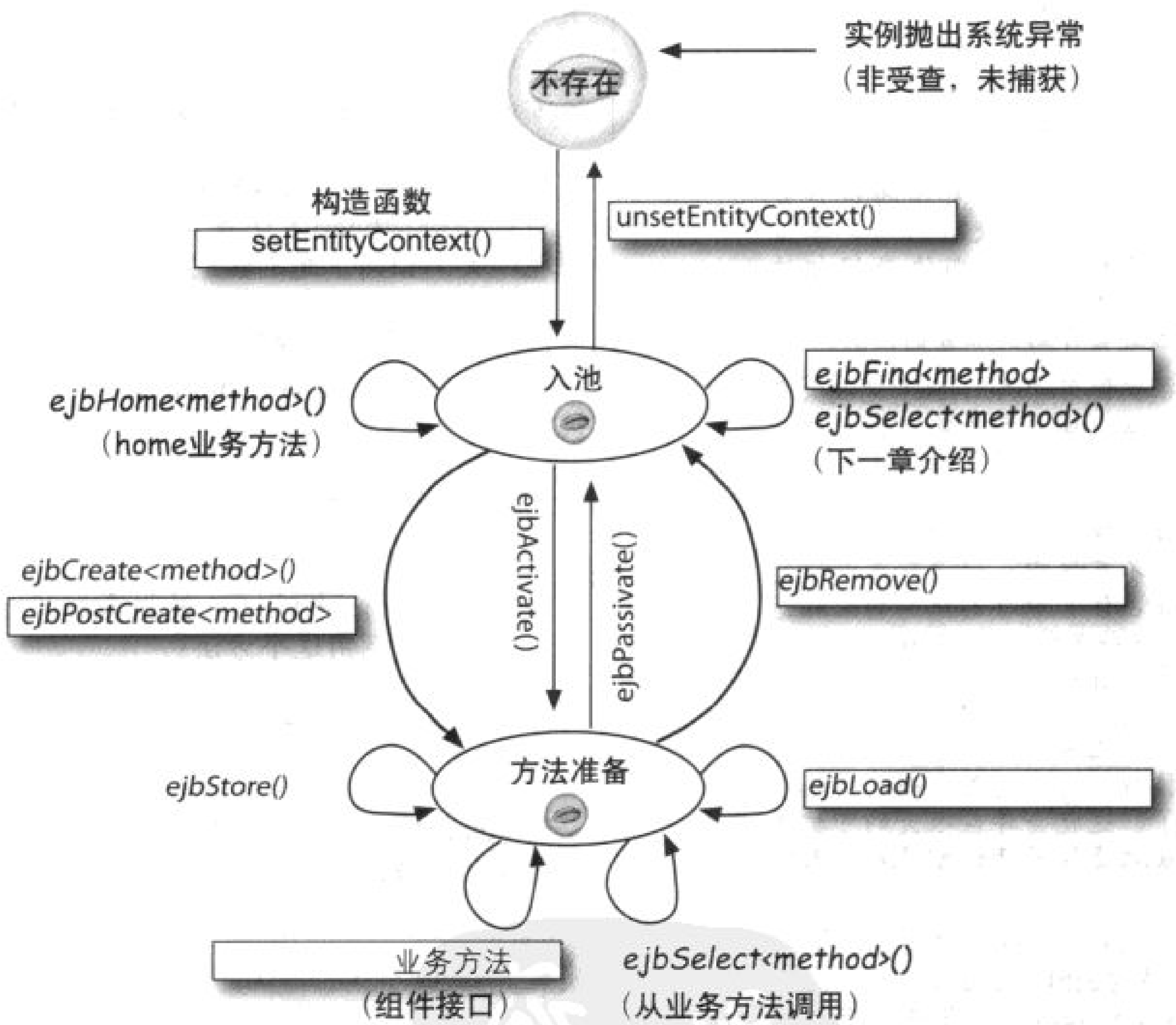
______________________

4. bean在池中，而且客户在组件接口上调用了一个业务方法时会调用这个方法（要求写出这种情况下调用的第一个方法）。

______________________

## Sharpen your pencil

填空，填上少了哪些方法。如果方法名写得不对没有关系，只要知道在实体bean状态图中各个状态变迁上发生了什么就行。




**必须知道这个生命周期的每一个细节！**

考试要求你知道bean生命周期的每一个细节。正如我们前面所说的，最难的部分是，实体bean中尽管与会话bean中有同名的方法，但是与会话bean中的方法相比，这些方法的含义是完全不同的。请特别注意：

**创建方法**（建立一个新bean与在数据库建立一个新实体）

**删除方法**（告诉容器你已经做完了与删除数据库中的一个实体）

**钝化**［串行化有状态bean与把bean（整个）发回到池中］

**激活方法**（逆串行化有状态bean与把bean从池中取出运行一个业务方法）

（要记住，无状态会话bean在池中进进出出的时候没有方法得到调用）

# 客户怎样得到#28的EJB对象引用？

当客户想要在一个特定实体上调用一个业务方法时（也就是说，这是一个有惟一主键的实体，如Lee Loo #28），首先需要该实体EJB对象的一个引用。

对于实体bean，要记住，得到EJB对象的引用可能有3种情况：

**① 客户在home上调用一个查找方法**

```
customerHome.findByPrimaryKey("28");
```

**② 客户在home上调用创建方法（第一次插入#28）**

```
customerHome.create("Loo", "Lee","28", "54 Bar Circle");
```

**③ 客户一个home业务方法，这个方法返回bean组件接口的一个引用**

```
customerHome.getCustomerByStreet("Bar Circle");
```

我们会分别讨论bean生命周期中关于查找方法、创建方法和home业务方法的各个阶段。不过，这里首先有一个假设，就是bean实例已经在池中。但是，bean最早又是怎么到池里去的呢？




客户使用bean做任何事情之前（创建方法、查找方法、业务方法等等），容器必须在池中建立一个新的bean实例。bean的构造与实体bean的创建并没有绑在一起。

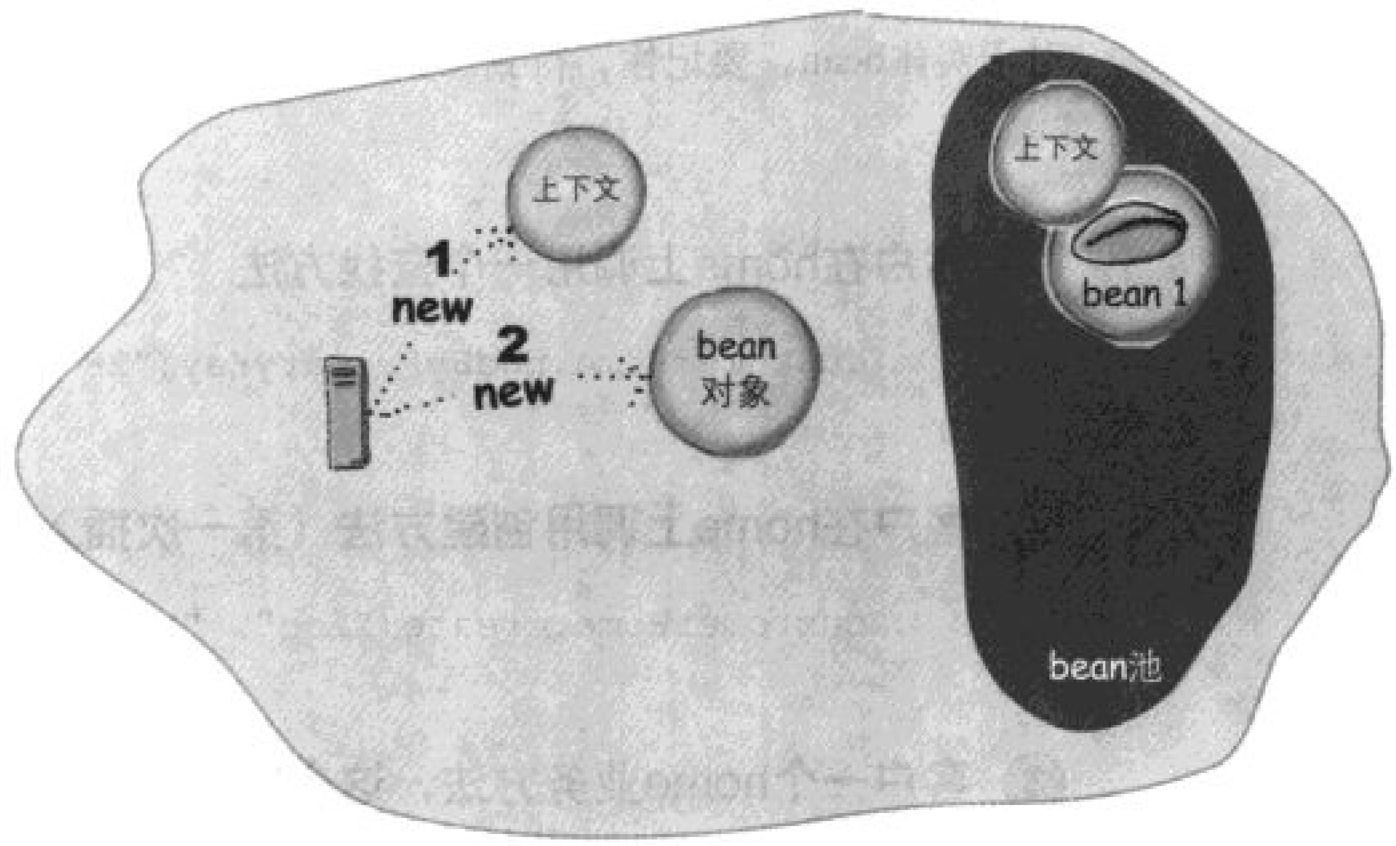
构造意味着一个新的bean实例。而创建则不同，对于一个实体bean，这意味着一个新的实体插入到数据库中。

# CMP实体bean构造

场景：容器想在池中建立一个新的bean实例

**1** 容器建立一个新的EntityContext

**2** 容器建立bean类的一个新实例（运行bean的构造函数）

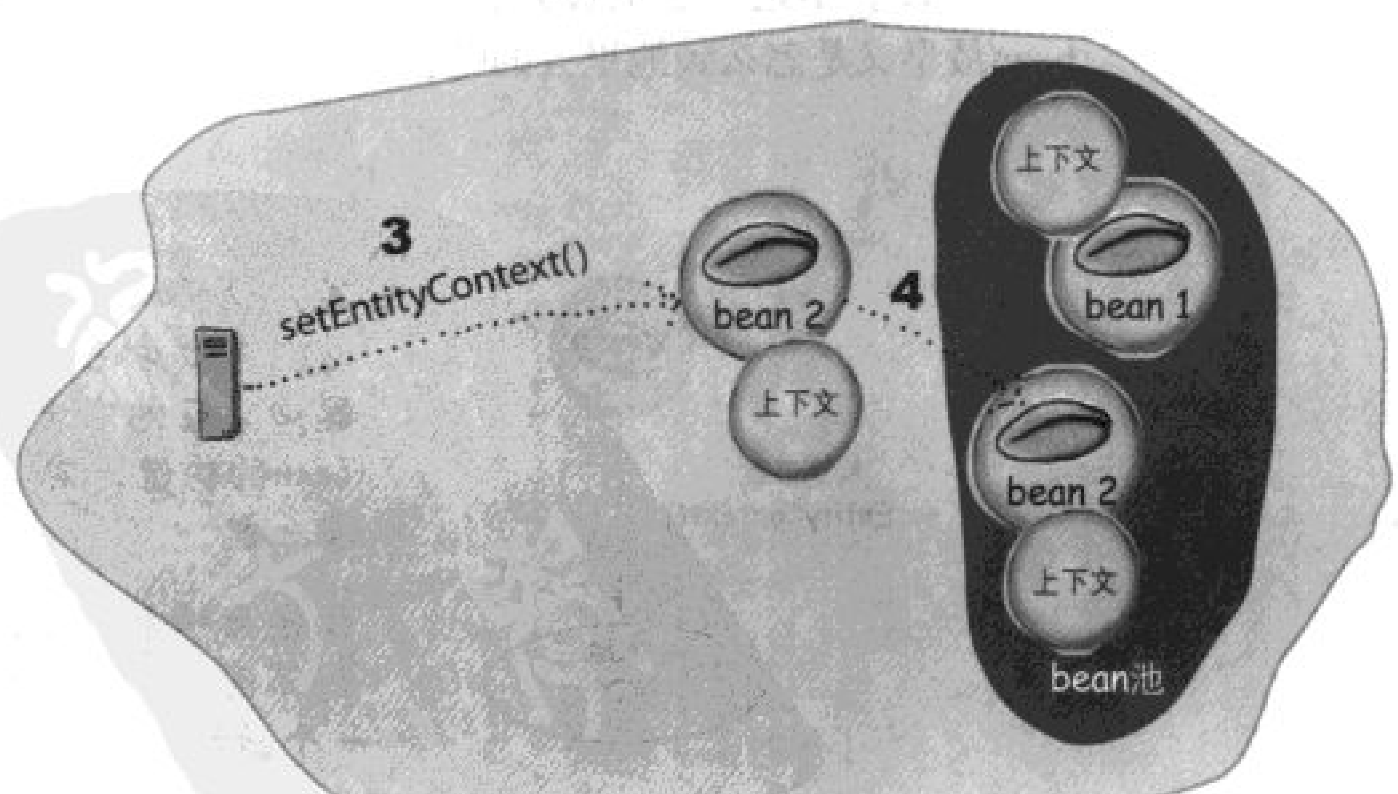



---

**3** 容器在bean上调用setEntityContext()。这是bean实例一生中惟一的一次调用这个方法。

**4** 容器把bean（现在有了一个上下文）放入池中。

注意：create()不会得到调用！create()方法只是用于向数据库中插入一个新实体，而与bean实例的创建无关！



# 实体构造期间可以做的bean工作：

时间 →

## 构造函数

使用 EntityContext可以：

- ❑ 得到home的一个引用
- ❑ 得到EJB对象的一个引用
- ❑ 得到有关客户的安全信息
- ❑ 要求事务回滚（CMT
- ❑ 查看事 … 回滚
- ❑ 得到一个事务引用，并在它之上调用方法（BMT bean）

**你什么也不能做**

可以访问：

- ❑ 特殊的JNDI环境
- ❑ 另一个bean的方法
- ❑ 资源管理器（如数据库）

## SetEntityContext()

使用 EntityContext可以：

- ☑ 得到home的一个引用
- ❑ 得到EJB对象的一个引用
- ❑ 得到有关客户的安全信息
- ❑ 要求事务回滚（CMT bean）
- ❑ 查看事务是否已经置为回滚（CMT bean）
- ❑ 得到一个事务引用，并在它之上调用方法（BMT bean）

可以访问：

- ☑ 特殊的JNDI环境
- ❑ 另一个bean的方法
- ❑ 资源管理器（如数据库）

### 构造函数里放什么

到目前为止我们知道，显而易见，构造函数里……什么也不放。

除非要求你这么做，否则甚至根本不要在你的代码中放构造函数，只使用编译器生成的默认构造函数就行了。

不管你做什么，都要确保你的类中有一个公共的（public）无参数构造函数！

```
public Customer() { }
```

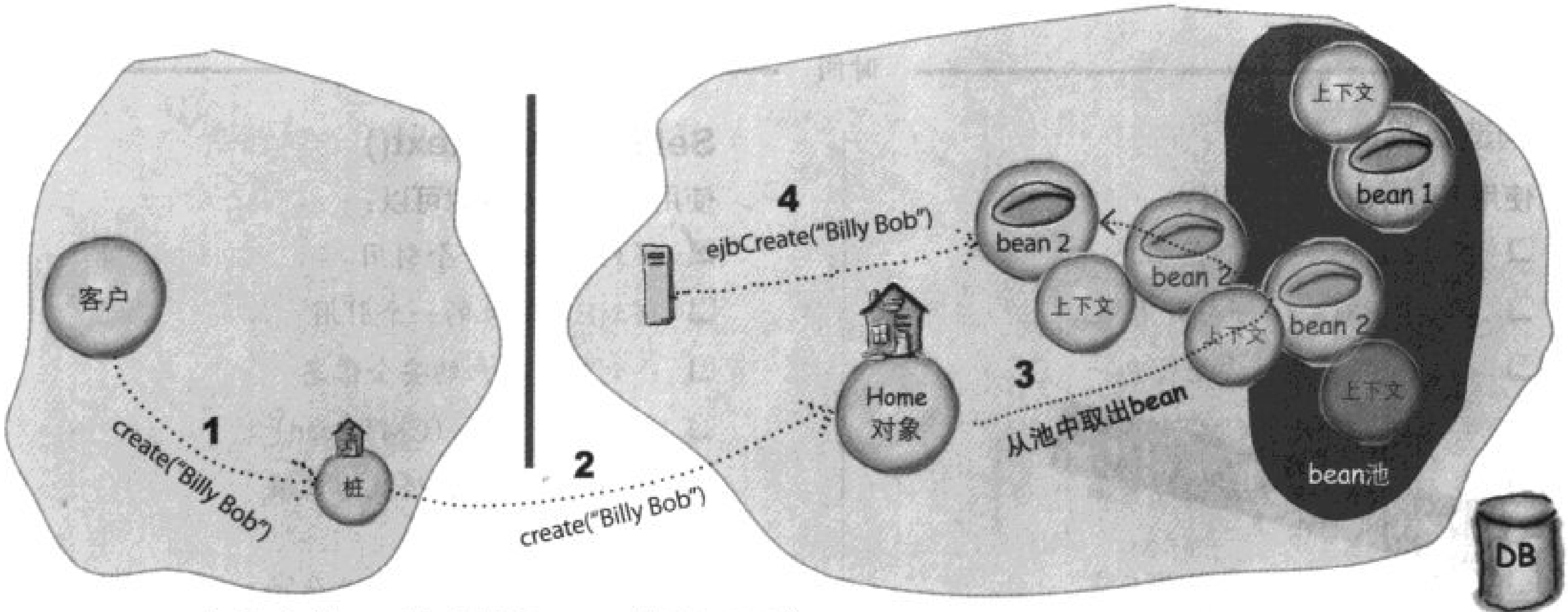
### setEntityContext()方法里放什么

把上下文赋给一个实例变量。要记住，你只有一次机会保存这个上下文。尽管并不总是要使用会话上下文，但是实体上下文可能是需要的。另外，不论你是否保留了实体上下文的一个引用，这个上下文都会保持活动，所以如果没有保存实体上下文，节省下来的只是引用变量的内存，对象本身所占的内存是省不下来的。

```
public void setEntityContext(EntityContext ctx) {
   context = ctx;
}
```

# CMP 实体bean创建

场景：客户想在数据库中创建一个新实体（记住，容器前面已经建立了bean实例和上下文，并把它们放到了池中）



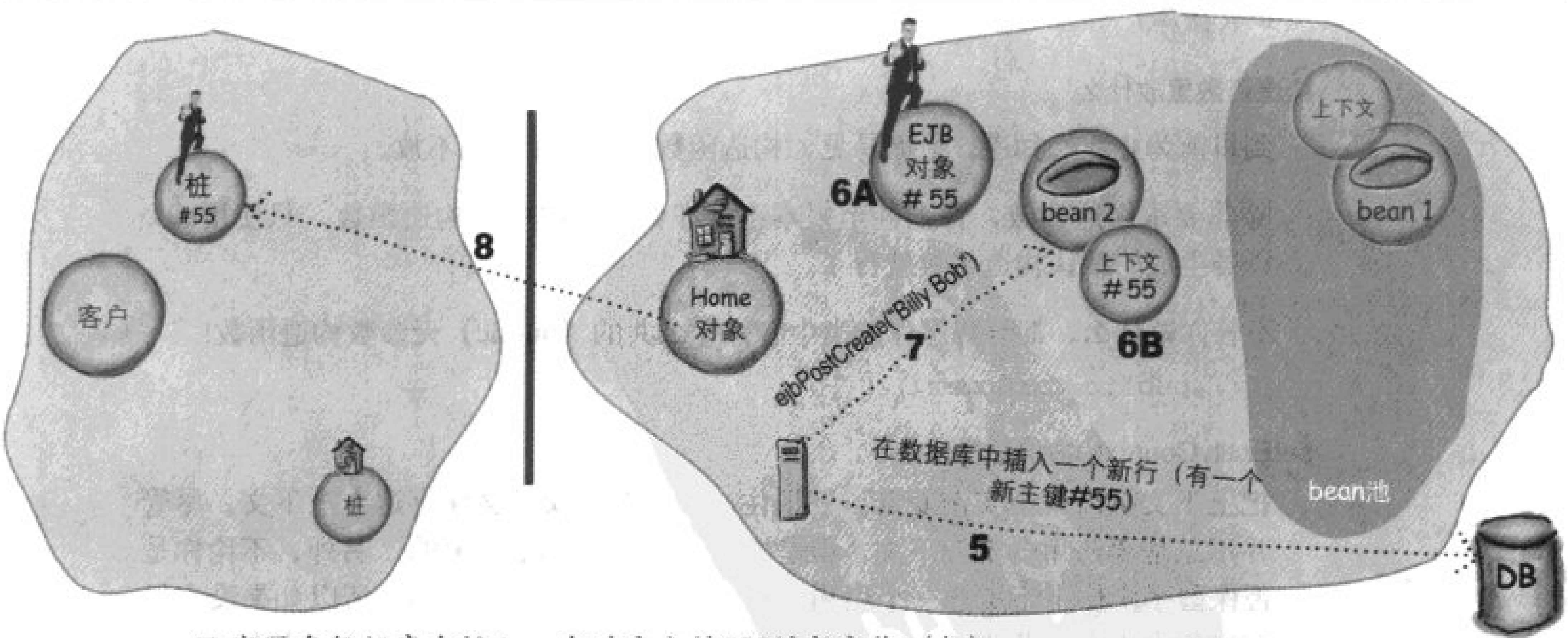

**1** 客户在home桩上调用create( "Billy Bob" )。

**2** create( "Billy Bob" ) 方法调用传递到home对象。

**3** 从池中取出一个bean来完成创建。

**4** 容器在bean实例上调用ejbCreate( "Billy Bob" )。

---




**5** 容器在数据库中插入一个对应主键#55的新实体（行）。

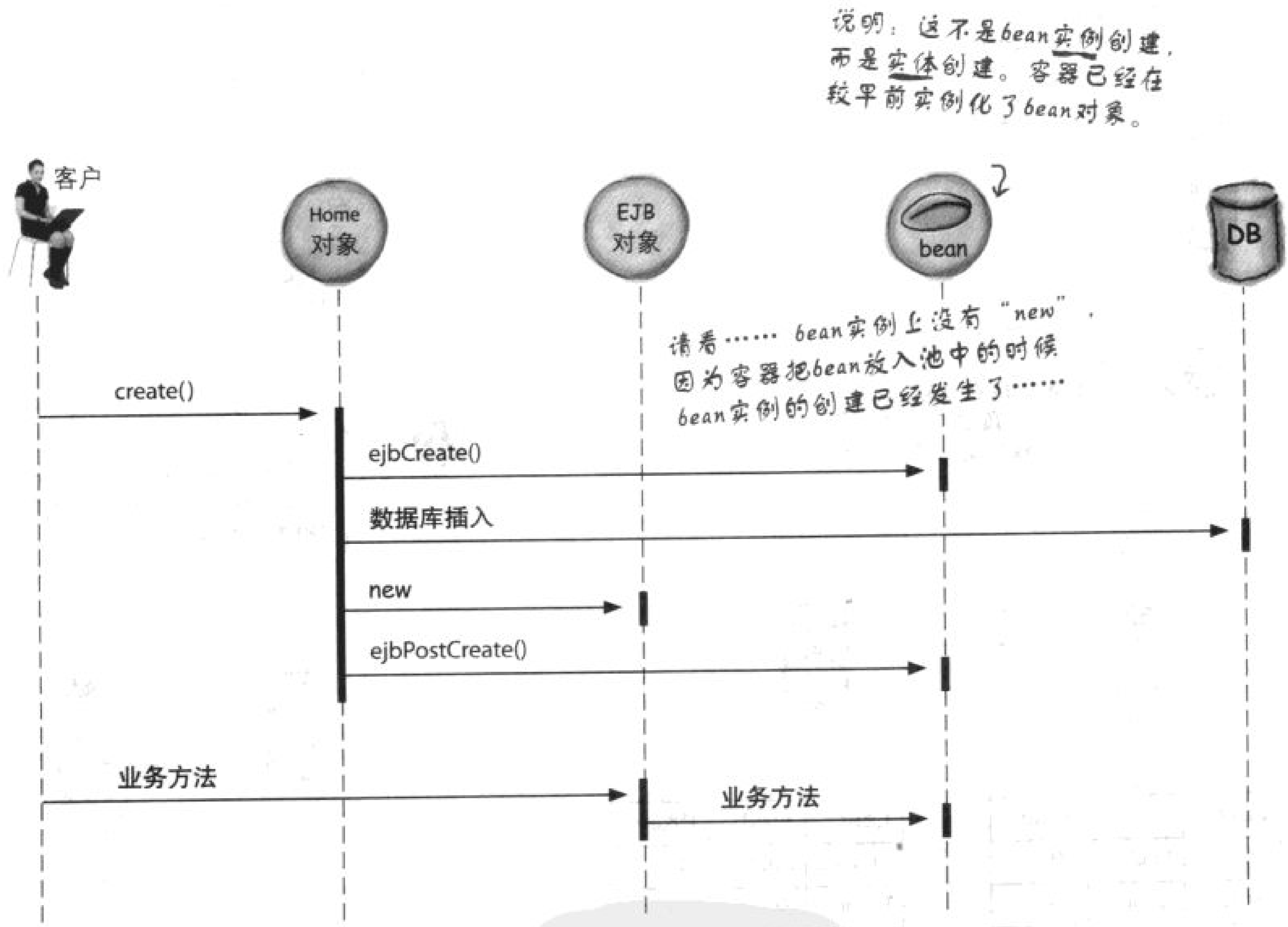
**6** 容器把这个主键值（#55）交给EJB对象和EntityContext。

**7** 容器在bean上调用ejbPostCreate( "Billy Bob" )，使bean有机会自行完成初始化。

**8** home把EJB对象桩返回给客户。

# 创建一个新实体

## CMP实体bean




**哇呜！真是与会话bean创建大相径庭……**

容器只在会话bean实例上调用一次ejbCreate(),也就是在bean一生的最初。这是会话bean和实体bean之间的一个很大的区别，实体bean可以反复地运行ejbCreate()，就像是运行一个业务方法一样。实际上，最好就这样来考虑ejbCreate()……把它想成是另外一个home业务方法，就好像方法名是ejbInsert()一样（在我们看来，就应该取这个名字，不过，规范小组又没有咨询我们的意见）。实体bean和会话beanejbCreate()方法的第二个区别是创建EJB对象的顺序。实体bean运行ejbCreate()时，并没有EJB对象！！没错！这意味着bean没办法得到它的保镖的一个引用。换句话说，实体bean无法使用ejbCreate()方法来做一些事情，比如说，得到它自己EJB对象的一个引用，并把这个引用传递给别人。要记住，bean不能传递自己的引用，绝对不行！如果一个bean想传递自己的一个引用，那么传递的必须它自己的EJB对象的一个引用。但即使是EJB对象的引用在ejbCreate()中也是无法传递的，因为此时还为时太早，根本没有EJB对象！

# CMP实体bean创建













之前

| Last | First | PKey |
|---|---|---|
| Frankie | Foof | 99 |
| Bryan | Boof | 314 |

之后

| Last | First | PKey |
|---|---|---|
| Frankie | Foof | 99 |
| Bryan | Boof | 314 |
| Billy | Bob | 55 |






# 实现你的ejbCreate()方法

## ejbCreate()里放什么

**这里要放实体初始化代码！这意味着要为你的持久字段设置值……持久字段就是映射到数据库中某些列的字段，更具体地，它们要映射到将插入数据库的行的某些列。**

要注意，我们没有说“把你的对象初始化代码放在这里……”，对于会话bean，我们曾经说，要把ejbCreate()当作是bean的一种构造函数来看待。但是对于实体bean，则是完全不同的，因为实例很早以前就已经初始化了（调用bean的setEntityContext()方法时就已经完成了实例化）。

所以，ejbCreate()是要把这个bean初始化为它现在表示的新实体。对于一个CMP bean，这意味着客户已经提供了有关的参数，这些参数表示了要插入到数据库中的数据。容器会查看你的持久字段的状态，一旦ejbCreate()返回，就会使用这些字段在数据库中建立新的行。

即使在ejbCreate()中没有做太多的事情，**你必须为这个实体建立一个主键**！这是最重要的任务。可以根据传递给创建方法的参数来建立主键。也可以基于一个键生成算法来建立主键。另外还可能有一个键服务器，你可以向它要下一个主键。不管采用什么办法，都必须提供一个主键，把这个值赋给主键字段。在执行ejbCreate()时，容器会查看持久字段的状态，并使用这些数据建立新的行。

警告！要记住，容器是在建立EJB对象之前调用ejbCreate()，所以不能使用ejbCreate()来得到你自己的引用（也就是你的保镖/EJB对象的一个引用）。在向上下文要你自己的组件接口的一个引用之前，必须等待ejbPostCreate()完成。而且不能使用ejbCreate()来访问你的持久关系；也是必须等着ejbPostCreate()完成（不过，这个内容会在下一章再做介绍）。

这必须是你的主键的类型！在这里，我们使用的主键是String，所以返回类型就必须是String（而且要在部署描述文件里声明这个类型）。

```
public String ejbCreate(String last, String first) {
    // 把参数赋给持久字段
    this.setLastName(last);
    this.setFirstName(first);

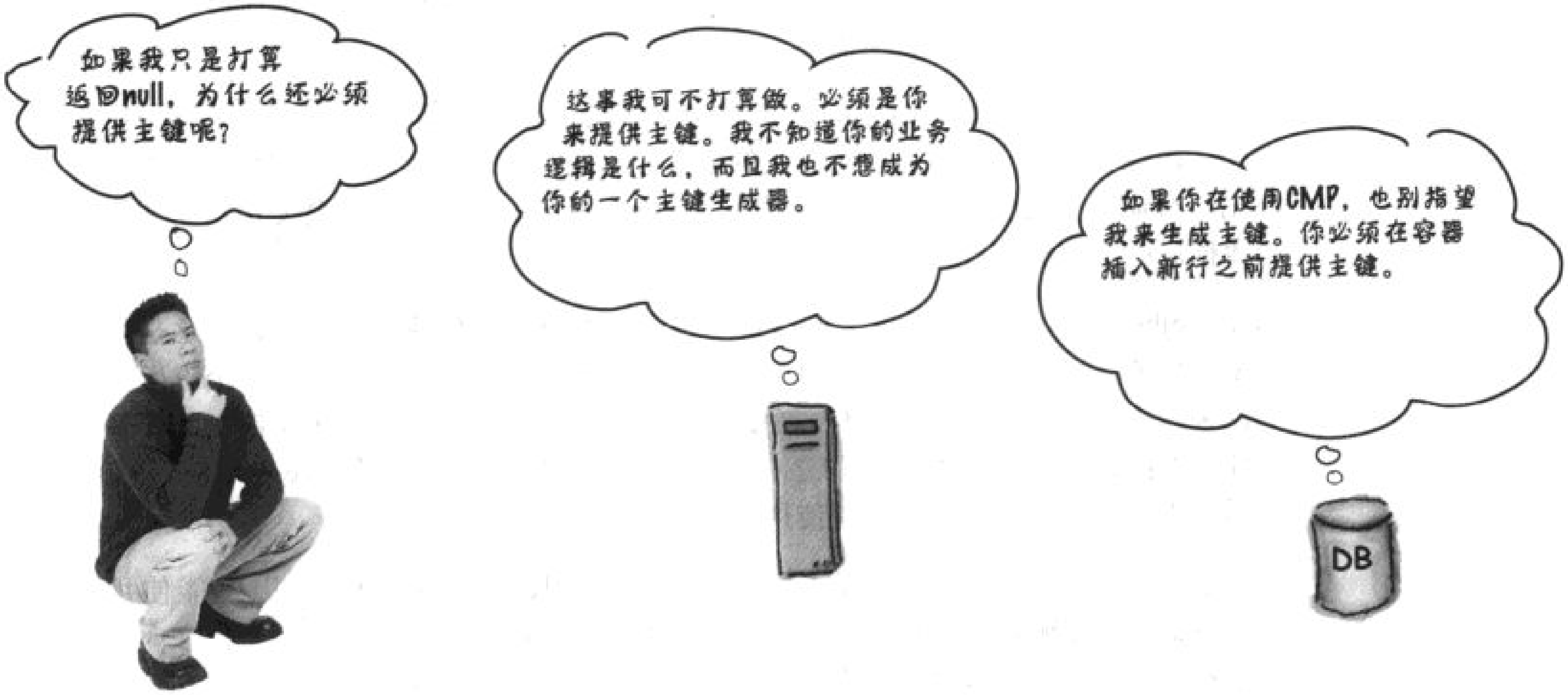
    // 设置一个主键
    this.setPrimaryKey(this.makeKey());
    return null;
}
```

一旦ejbCreate()完成，容器会使用你设置的这些值来建立新行。

返回null？？这是什么意思？

# 对象身份：主键

**每个实体都必须有一个惟一的身份。**容器不允许两个或多个实体有同样的主键。对于EJB 2.0，提供主键仍然是你的任务。容器不会做这个工作（至少根据规范，这个工作不是容器来做）。




**对于CMP，还是由你负责提供主键。**这并不是说不能使用某种主键服务。也许你有一个服务能自动地从数据库分配一大块主键，然后根据需要分配这些主键。或者，也可以有某种惟一标识符算法，能够为你建立惟一的主键。还可以使用顾客的社会保险号或帐号作为主键，这完全取决于你如何决定。

你怎么告诉容器这个将创建的实体主键值是什么？这要通过一个或多个持久字段的值。

如果只是把一个字段作为主键，而它直接映射至数据库中的一列，那就仅此而已，万事大吉了。但是，如果需要多个值来惟一地标识一个实体（比如，可能要取名字和日期的组合来标识），则可以使用一个复合主键，这个主键使用了一个或多个容器托管持久存储字段。

### ejbCreate()的规则

- 对home接口中的每个create<method>，必须实现一个匹配的ejbCreate<method>。
- 方法名必须以前缀“ejbCreate”开头。
- 方法必须声明为公共的（public），而且不能声明为static或final。
- 声明的返回类型必须是实体bean的主键类型，就算是这个方法要返回null也不例外。
- 方法参数必须与匹配的create<method>的参数相同。
- 可以声明一个抛出CreateException的throws子句，或者是你想抛出的任何其他应用（受查）异常，只要home接口中也声明了这个异常，不过，不能声明RemoteException。

在ejbCreate()的最后，必须将一个合法的主键赋给一个CMP字段（你已经告诉容器这就是你的主键字段）。

如果使用一个复合主键，那么构成这个主键的所有字段都必须有合法的值。

如果在ejbCreate()的最后，你的主键为null，容器就不会完成创建！

### 主键的规则

- 主键类必须是Serializable，而且必须是公共的（public）。
- 可以使用bean类的一个持久字段作为主键，这要在部署描述文件中指定字段名和类的类型。
- 如果需要两个或多个持久字段来惟一标识你的实体，就要建立一个定制的复合主键类。
- 复合主键类必须由bean类中定义为持久字段的字段组成。bean类中的字段必须有公共的存取方法。

**问：** **为什么不能让数据库提供主键？我们通常不都是这样做的吗？你是不是说容器不能做这个工作？**

**答：** 没错，这正是我们先前告诉你的。你与容器之间的合约宣称，等到ejbCreate()完成，你已经做了所有必要的工作来建立一个合法的主键。也许会有这样一个服务器，你可以对它说“你在完成插入的时候让服务器提供主键”，但是规范中对于这一点没有保证，而且不能假设所有EJB开发商都对此提供支持。也许在将来的某一天，可以用一种独立于部署的方式在部署描述文件中指出“使用自动生成的主键”，但是现在还不是这样。而且EJB 2.0规范还达不到这种水平。

# 实现你的ejbPostCreate()方法

## ejbPostCreate()里放什么

在这里完成你的初始化代码！现在可以了解到，为什么实体bean需要一个ejbPostCreate()方法了，这是因为ejbCreate()为时太早，还不能做很多事情。使用ejbPostCreate()有两个最重要的原因，其一是可以用它来得到你自己的EJB对象的一个引用，其二，必须用它来访问你的容器托管关系（container-managed relationship，CMR）（下一章将介绍CMR）。

可以把ejbPostCreate()想成是ejbCreate()的另一半，在这期间发生了一些相当重要的事情。如果需要做一些事情作为创建新实体的一部分，而且这些工作要以访问你的EJB对象为前提（或者其他只能在ejbPostCreate()中发生，而不能在ejbCreate()中发生的事情），那么这是第二次（也是最后一次）完成初始化的机会。

下一章介绍CMR的时候还会再谈到ejbPostCreate()。

## ejbPostCreate()的规则

- 对home接口中的每个create<method>，必须实现一个匹配的ejbPostCreate<method>。
- 方法名必须以前缀“ejbPostCreate”开头。
- 方法必须声明为公共的（public），而且不能声明为static或final。
- 声明的返回类型必须为void!
- 方法参数必须与匹配的create<method>的参数相同。
- 可以声明一个抛出CreateException的throws子句，或者是你想抛出的任何其他应用（受查）异常，只要home接口中也声明了这个异常，不过，不能声明RemoteException。

## 这3个方法的参数都相同，但是返回类型不同！

Home interface
create<method>

```
public Customer create(String last, String first)throws
                                    CreateException,RemoteException);
```

注意名字上的差别

Bean class
ejbCreate<method>

```
public String ejbCreate(String last, String first) {...}
```

另一个差别

Bean class
ejbPostCreate<method>

```
public void ejbPostCreate(String last, String first){...}
```

bean类不必声明同样的异常，除非确实抛出了这些异常（而且绝对不能从bean抛出一个RemoteException）。只有容器才能抛出RemoteException。

### BRAIN POWER

请考虑上面的3个方法，它们之间有什么关系。想一想每个方法各自负责什么。再来回答这两个问题：

1. 为什么要如此声明返回类型？换句话说，请解释这三个方法为什么声明了3个不同的返回类型……

2. 为什么ejbCreate<method>和ejbPostCreate<method>有相同的参数？


我实在想不通为什么一个bean需要它自己的EJB对象的一个引用……
假设你在数据库中放入一个新的Customer。也许你的业务逻辑指出，"建立一个新Customer时，要把这个新Customer实体bean的引用传递给CustomerRegistration bean，来通知市场部。"
我还是不明白为什么我需要EJB对象……我不能把"this"作为参数传给其他bean吗？
想想看。首先，这违反了bean法则。EJB规范不允许把"this"作为参数在bean代码中传递。除了容器之外，任何人都不能得到bean的一个直接引用！如果有人确实得到了bean的一个引用，那他只能在这个引用上单纯地调用方法，而无法得到我的任何服务！













不是这样：

```
doStuff(this);
```

要这样：

```
doStuff(myContext.getEJBObject());
```



## 实体创建期间可以做的bean工作：

时间 →

### ejbCreate()

*使用 EntityContext可以：*

- ☑ 得到home的一个引用
- ☐ 得到EJB对象的一个引用
- ☐ 得到主键
- ☑ 得到有关客户的安全信息
- ☑ 要求事务回滚（CMT bean）
- ☑ 查看事务是否已经置为回滚（CMT bean）
- ☐ 得到一个事务引用，并在它之上调用方法（仅适用于BMP bean，所以实体不能使用这个事务引用）

*可以访问：*

- ☑ 特殊的JNDI环境
- ☑ 另一个bean的方法
- ☑ 资源管理器（如数据库）

要得到EJB对象的一个引用或者主键，还为时
太早，因为容器还在等着你完成ejbCreate()。
一旦你从ejbCreate()返回，容器会查看bean，看
你基于客户提供的参数做了初始化之后会修
改哪些状态。只有这个时候容器才能知道你
的主键是什么（通过查看主键字段！），而
且它需要这个主键信息来建立一个EJB对象。

### ejbPostCreate()

*使用 EntityContext可以：*

- ☑ 得到home的一个引用
- ☑ 得到EJB对象的一个引用
- ☑ 得到主键
- ☑ 得到有关客户的安全信息
- ☑ 要求事务回滚（CMT bean）
- ☑ 查看事务是否已经置为回滚（CMT bean）
- ☐ 得到一个事务引用，并在它之上调用方法（仅适用于BMP bean，所以实体不能使用这个事务引用）

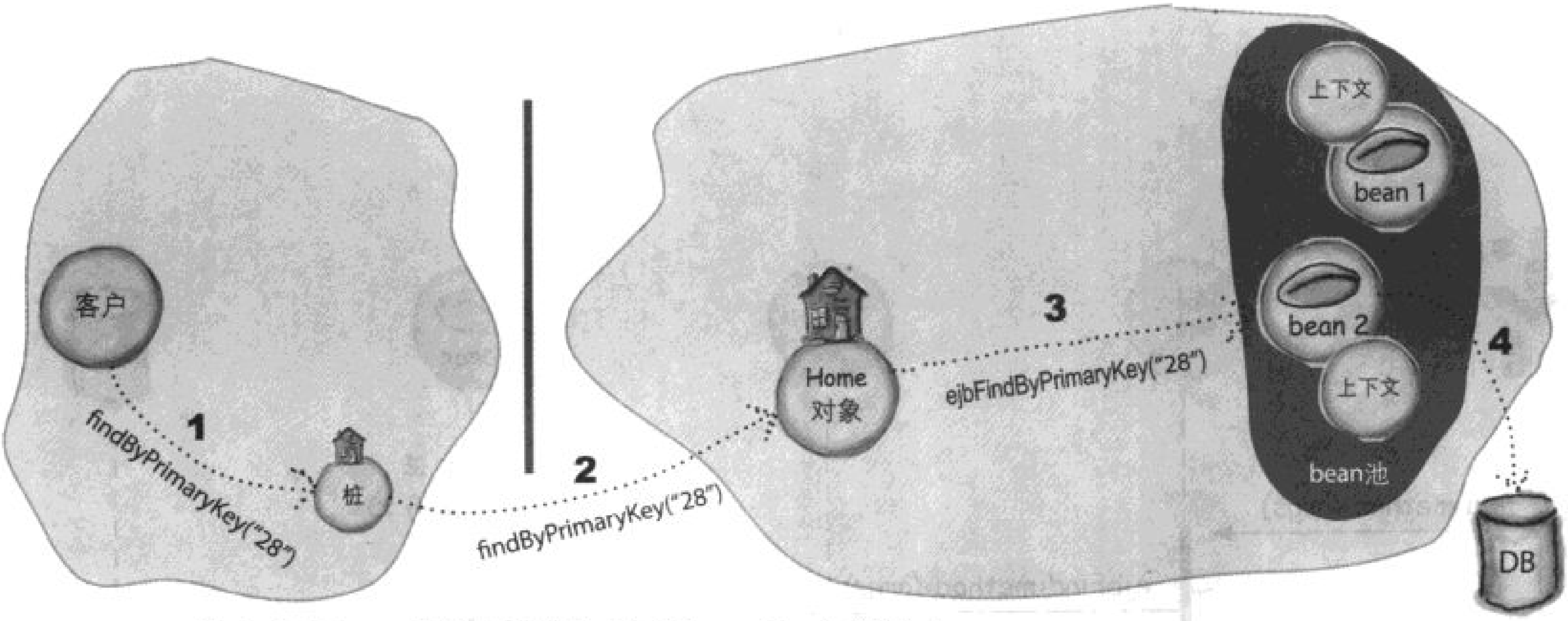
*可以访问：*

- ☑ 特殊的JNDI环境
- ☑ 另一个bean的方法
- ☑ 资源管理器（如数据库）

到执行ejbPostCreate()的时候，容
器已经知道你的主键，而且建
立（或找到）了对应这个主键的
EJB对象。

# CMP实体查找方法

场景：客户想得到一个现有实体bean的引用

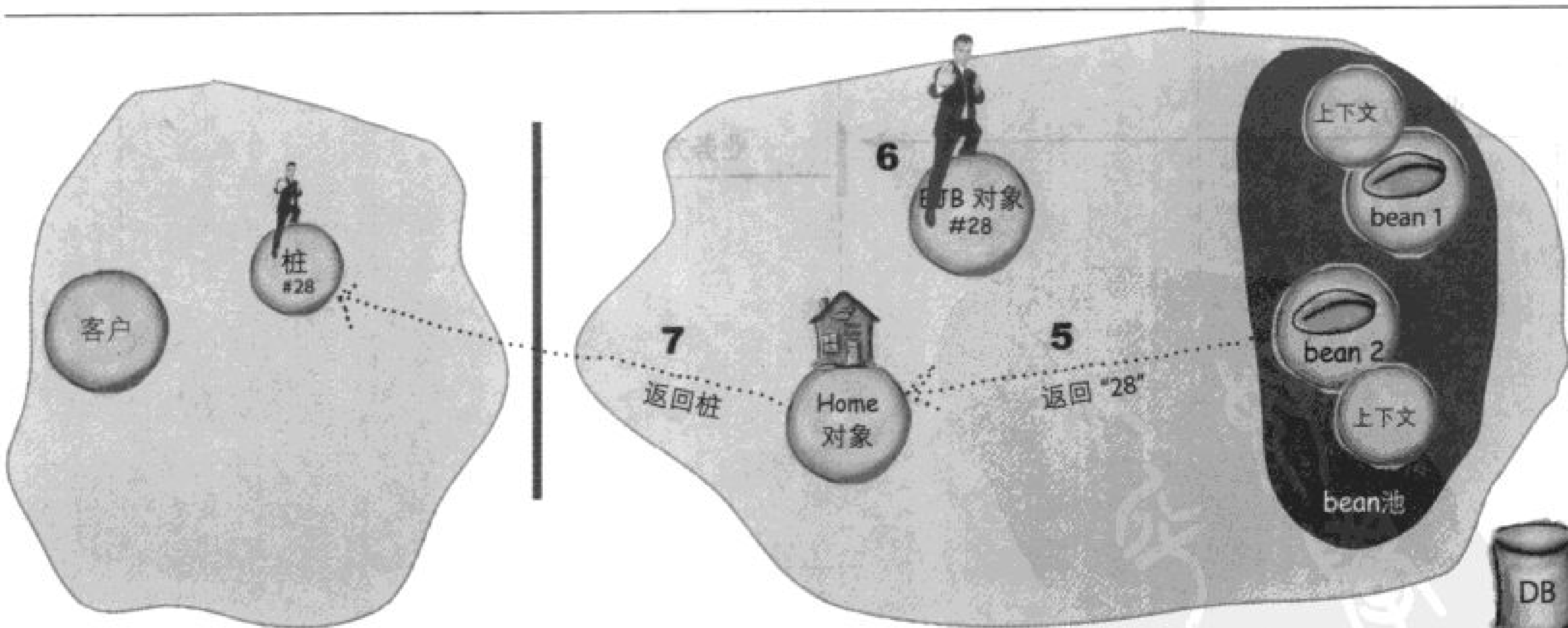



**1** 客户在home引用上调用findByPrimaryKey(“28”)。

**2** 这个查找方法传递给home对象。

**3** 从池中选出一个bean来运行ejbFindByPrimaryKey(“28”)方法。

**4** 这个bean在数据库上完成一个选择操作，验证得否存在主键为#28的一个实体。




**5** 这个bean把主键(#28)返回给home，这说明数据库中存在这样一个实体。

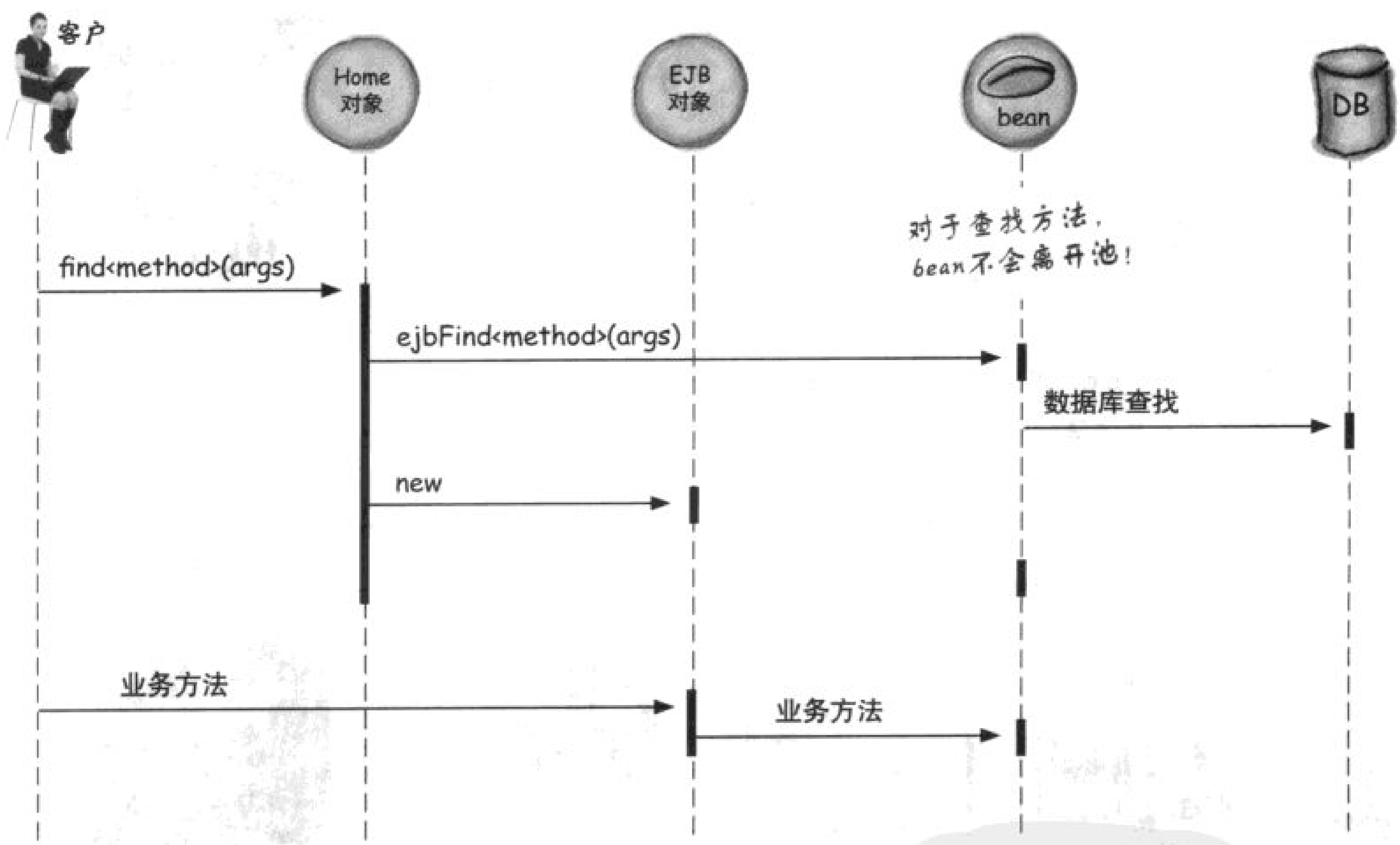
**6** 容器建立（或找到）对应#28的一个EJB对象。

**7** 容器返回对应#28的EJB对象桩。

说明：如果bean没有找到一个主键为#28的匹配实体，客户就会得到一个ObjectNotFoundException异常（这是FinderException的一个子类）。

## 查找一个现有的实体

CMP实体bean

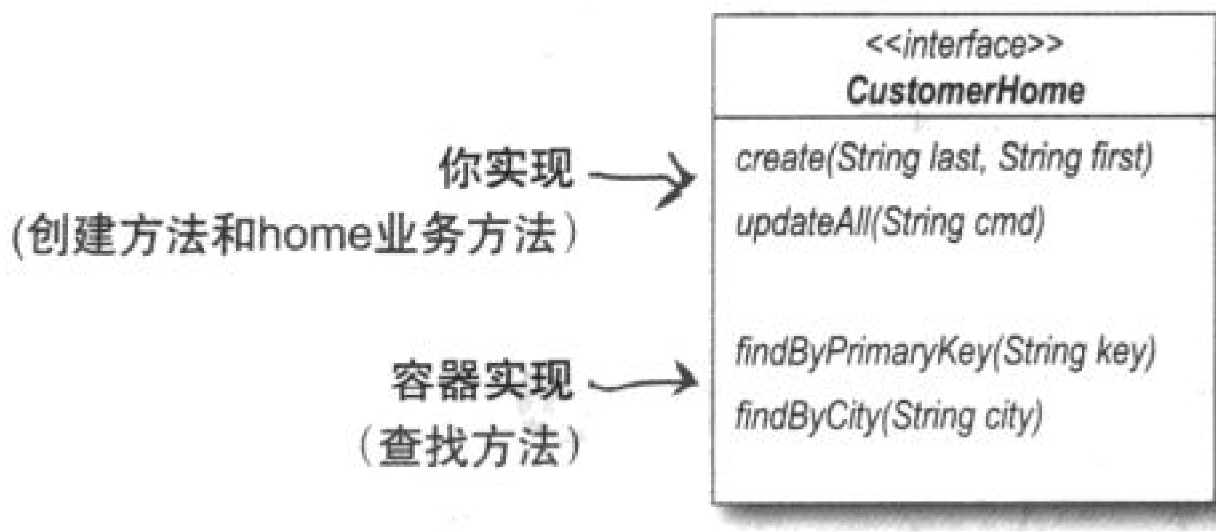


# CMP实体bean查找方法




| Last | First | PKey |
|---|---|---|
| Poly | Morphism | 72 |
| Dewey | Cheatem | 900 |

# 你不用实现查找方法！



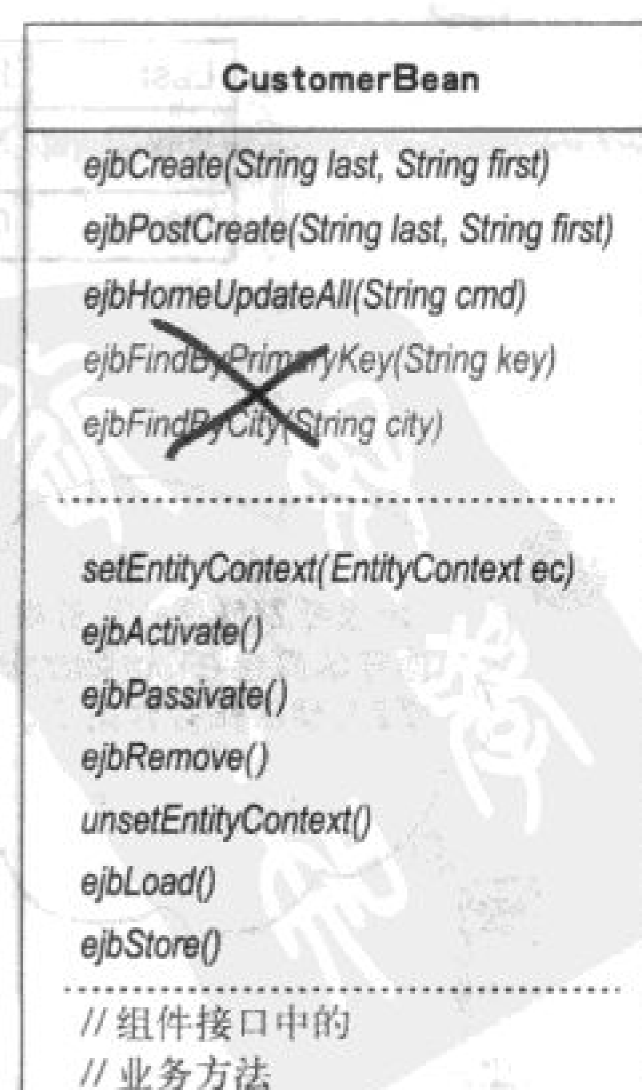

不要在bean类中放任何有关查找方法的东西。容器会使用home接口和部署描述文件来得出该做什么，并实现你的查找方法。

bean类甚至不能提到查找方法。

根本不用写查找方法。对于一个CMP bean，不要在bean类中放任何有关查找方法的代码。你只要实现创建方法和home业务方法就行了，对你的bean类而言（也就是你所写的东西），根本没有查找方法。

当然，要在home接口中声明这些查找方法，包括必须要有的findByPrimaryKey()，但是不用编写有关查找方法的任何其他Java代码。

容器会查看你的home接口和部署描述文件，得出怎样实现查找方法，然后容器会编写所有查找方法的代码。你是不会看到这些的。




嗯……我注意到bean没有从池中出来。不过，最让人奇怪的是，现在有了一个对应#72的EJB对象，却没有扮演#72的实体bean。没有用数据库中的实体数据加载bean，那么，如果客户调用了一个业务方法（比如getAddress()），会发生什么事情呢???
客户有可能根本不在那个EJB对象上调用业务方法，在这种情况下，我为什么要用数据来加载bean呢？这样只会浪费bean的时间。
你说的我知道了，但是……客户还是有可能在bean上调用业务方法的，对吧？这样他们不就得费劲地去找一个吗？
你说的没错……确实，这个问题让我想想。客户是有可能在实体上调用一个业务方法，所以看上去让bean准备好似乎更高效。但是，即使我在查找方法期间确实在bean中加载了数据，我还是得重新加载。为什么呢？因为等到客户调用业务方法的时候，bean的状态可能已经过时了！

我没听太懂……你说过时是什么意思？如果别人都不能得到bean的数据，bean怎么会过时呢？其他人怎么修改实体？

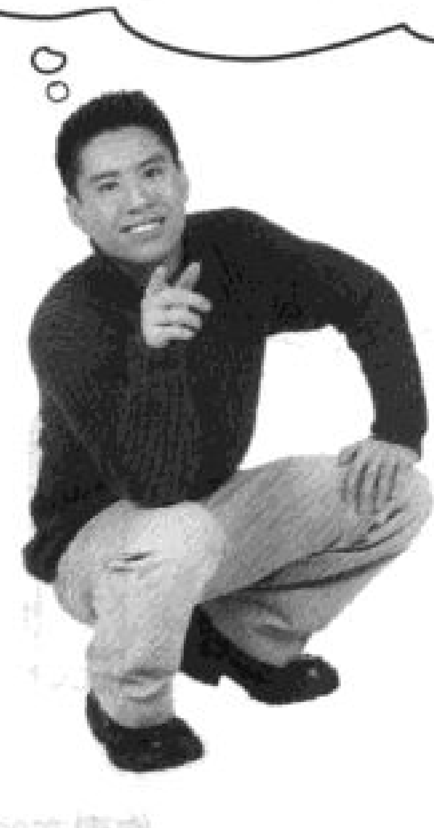


想想看。实体（比如说Poly Morphism）生活在数据库里（这是表中的一行）。当然，我们可以把这个实体的EJB对象的引用发出去，但是……实体bean并不是访问数据库的惟一途径！所以在客户等着调用一个方法时，我们知道，别人可能已经更新了数据库中Poly的记录。

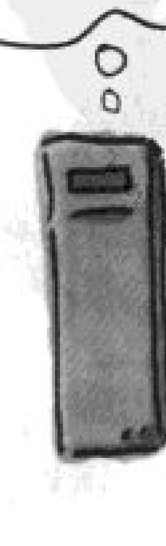
如果是这样，那么当客户在Poly实体bean上调用方法的时候，bean只会返回原来的错误的信息。



噢，我明白了……如果在查找方法期间用数据加载了bean，等到客户在这个bean上调用业务方法的时候，数据库中的底层实体数据可能已经改变了。

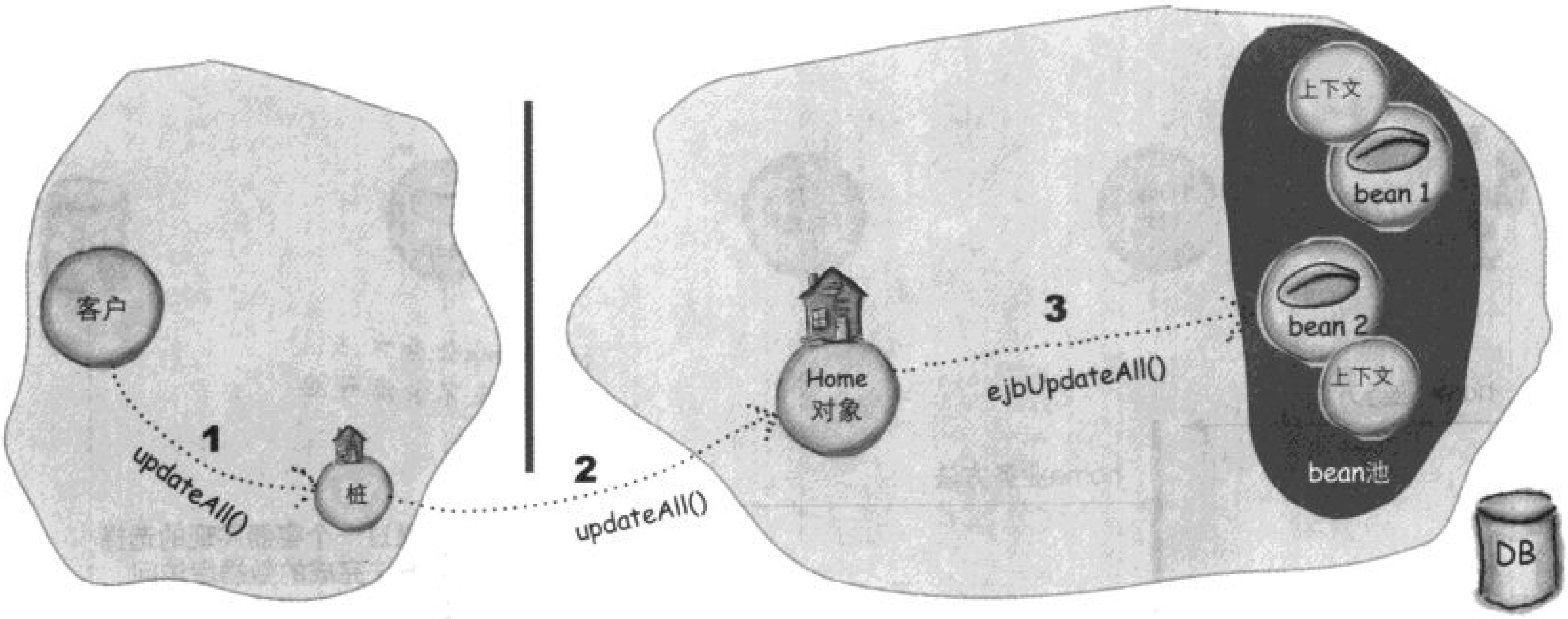


正是如此!这样的话，客户一调用业务方法，我就必须再次加载数据，来确保已经用最新的当前数据刷新了bean，所以在查找期间加载bean毫无意义。这只是一种资源浪费。



# CMP Home业务方法

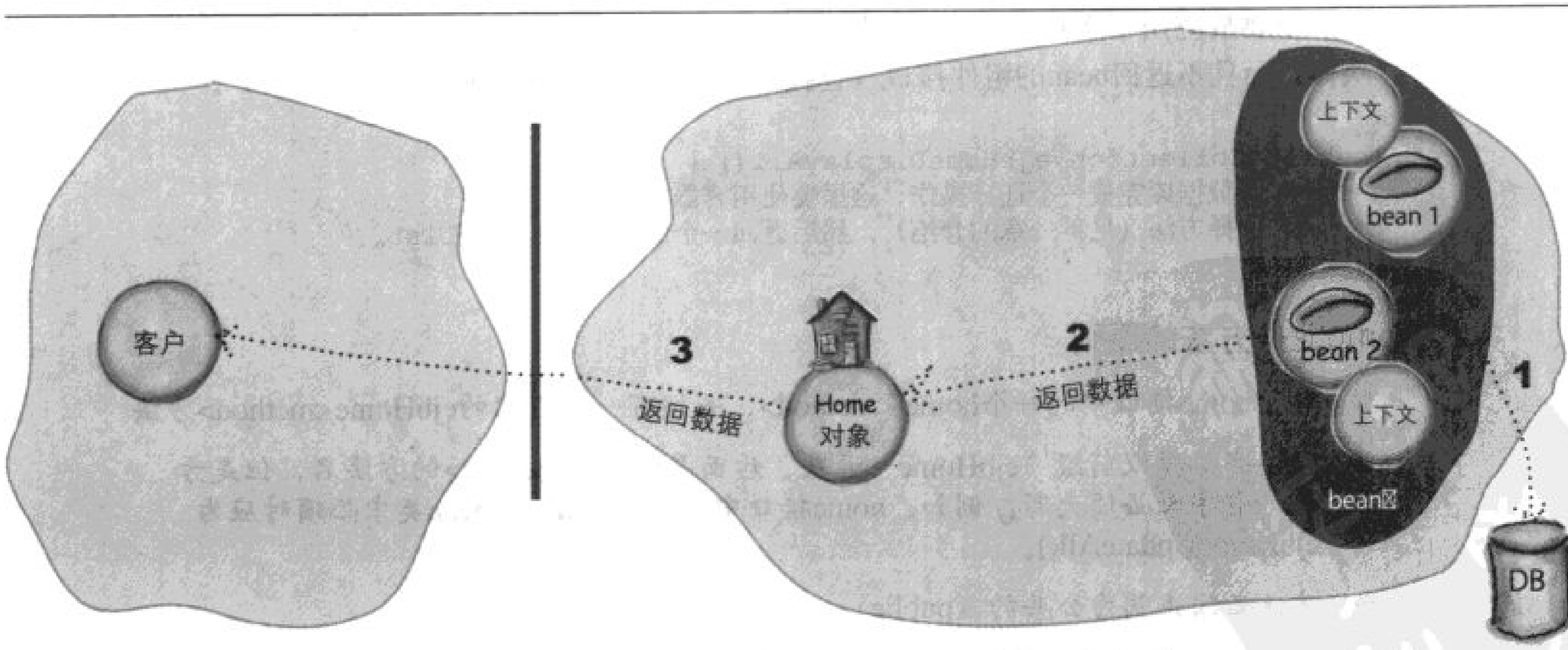
场景：客户想调用home中的一个业务方法




**1** 客户在home引用上调用updateAll()。

**2** updateAll()方法传递到home对象。

**3** 从池中选出一个bean来运行ejbUpdateAll()方法。



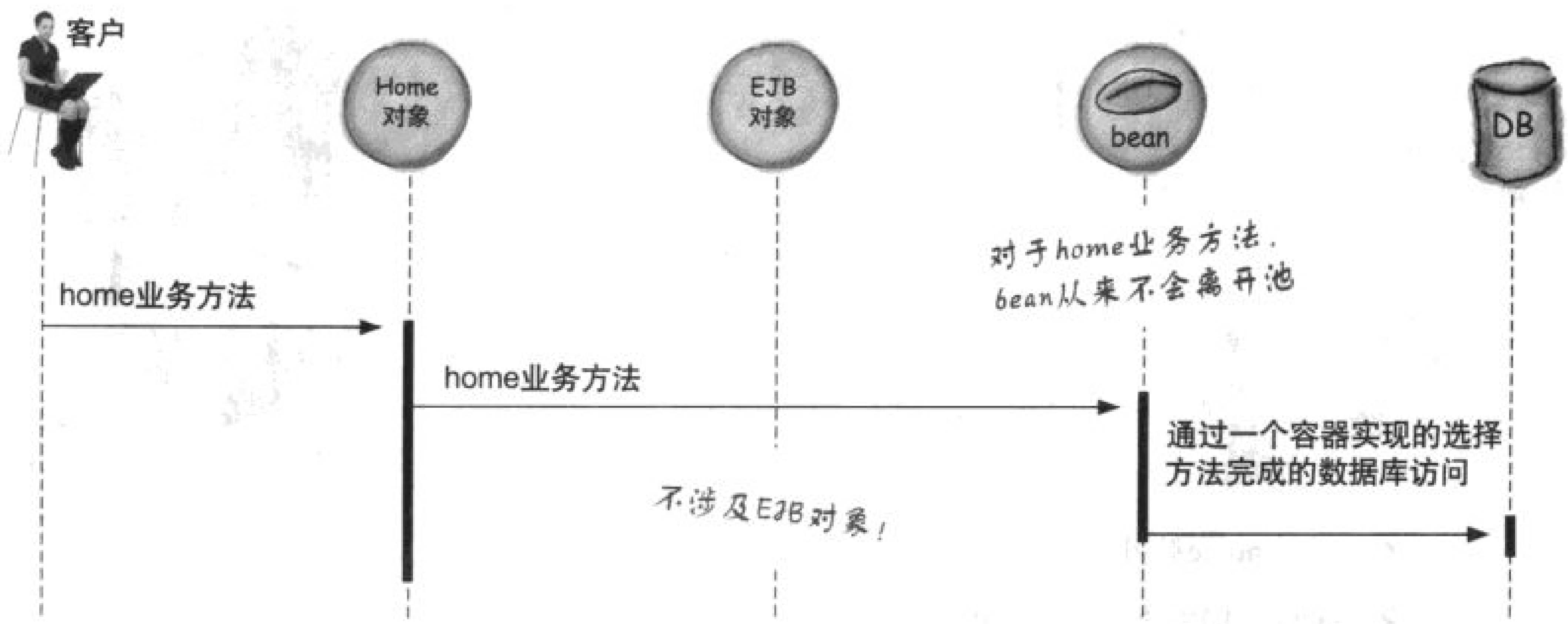

**1** bean没有离开池就会运行home方法，可能会调用一个容器实现的select<method>方法来访问数据库。

**2** bean从这个方法返回（可能会返回数据）。

**3** 容器把返回值传回给客户。

# Client调用一个Home业务方法

## CMP实体bean




### home业务方法中放什么

在编写代码时，要记住，实体bean运行home业务方法的时候是待在池里的。在home业务方法中要放应用于底层数据库的代码，这些代码针对的是一组实体，而不是一个特定的实体，而且不返回bean的组件接口（查找方法和创建方法则必须返回组件接口）。

```
public Collection ejbHomeDisplayAll() {
   // 对数据库完成一个选择操作，这里要使用容器实现的
   // 选择方法（见下一章的介绍），然后返回一个String ArrayList。
```

### home业务方法的原则

- 对于home接口中的每个home <method>，必须有一个匹配的ejbHome<method>方法。
- 方法名必须以前缀“ejbHome”开头，后面是home <method>的方法名，但是方法的首字母必须大写。例如，home接口中的updateAll()在bean类中必须对应为ejbHomeUpdateAll()。
- 方法必须声明为公共的（public），而且不能声明为static。

  如果home接口是远程的，参数和返回值都必须是合法的RMI-IIOP类型。
- 可以声明抛出你自己的应用（受查）异常的throws子句，只要home接口中也声明了这些异常。
- 无论如何，都不能声明RemoteException。

# home业务方法中可以做的bean工作

**使用 EntityContext可以：**

- [x] 得到home的一个引用
- [ ] 得到EJB对象的一个引用
- [ ] 得到主键
- [x] 得到有关客户的安全信息
- [x] 要求事务回滚（CMT bean）
- [x] 查看事务是否已经置为回滚（CMT bean）
- [ ] 得到一个事务引用，并在它之上调用方法（仅适用于BMP bean，所以实体不能使用这个事务引用）

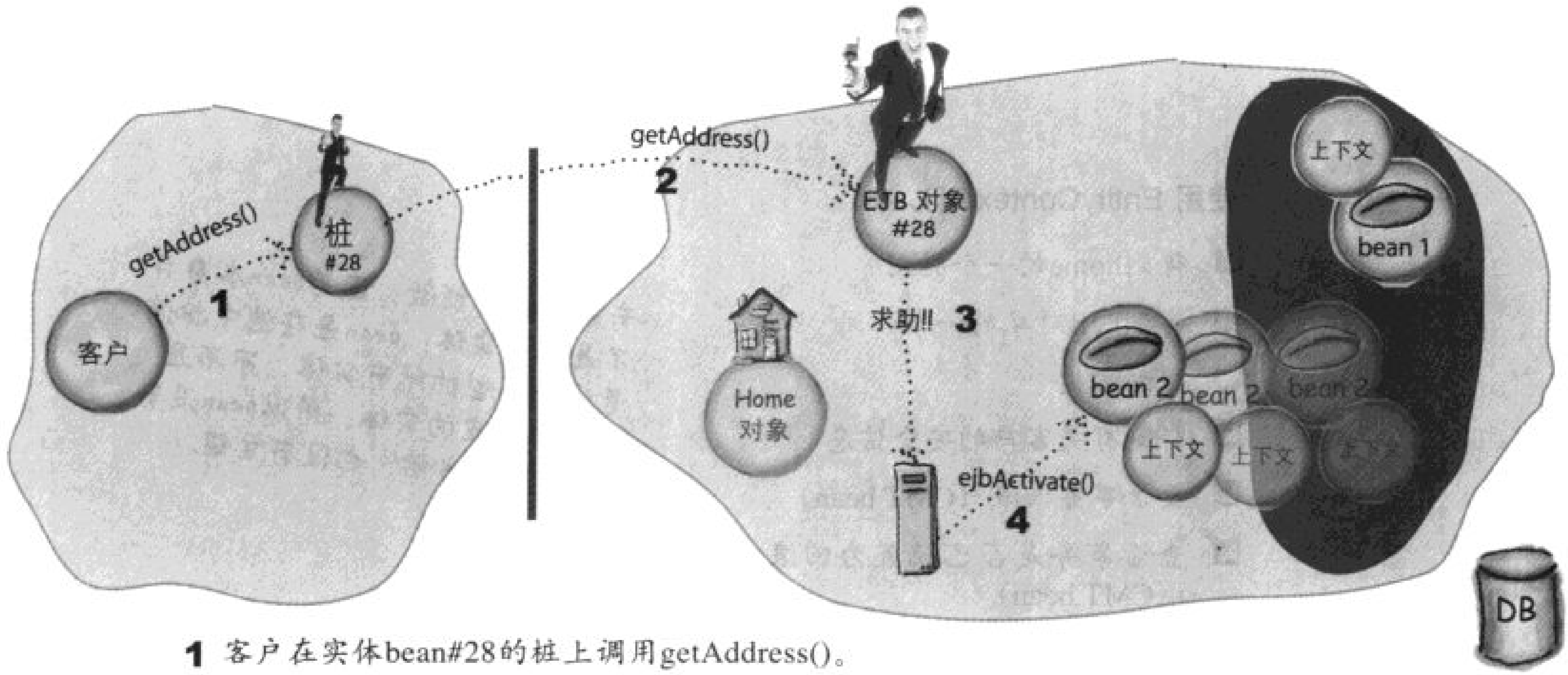
**可以访问：**

- [x] 特殊的JNDI环境
- [x] 另一个bean的方法
- [x] 资源管理器（如数据库）

有两件事不能做，因为bean本身并不是一个实体！bean是在池中扮演着这一类型的所有实体，而不是某一个特定的实体。所以bean没有EJB对象，当然，也没有主键。

# 开始一个业务方法（在事务中）

场景：客户想得到一个特定CMP Customer实体的地址



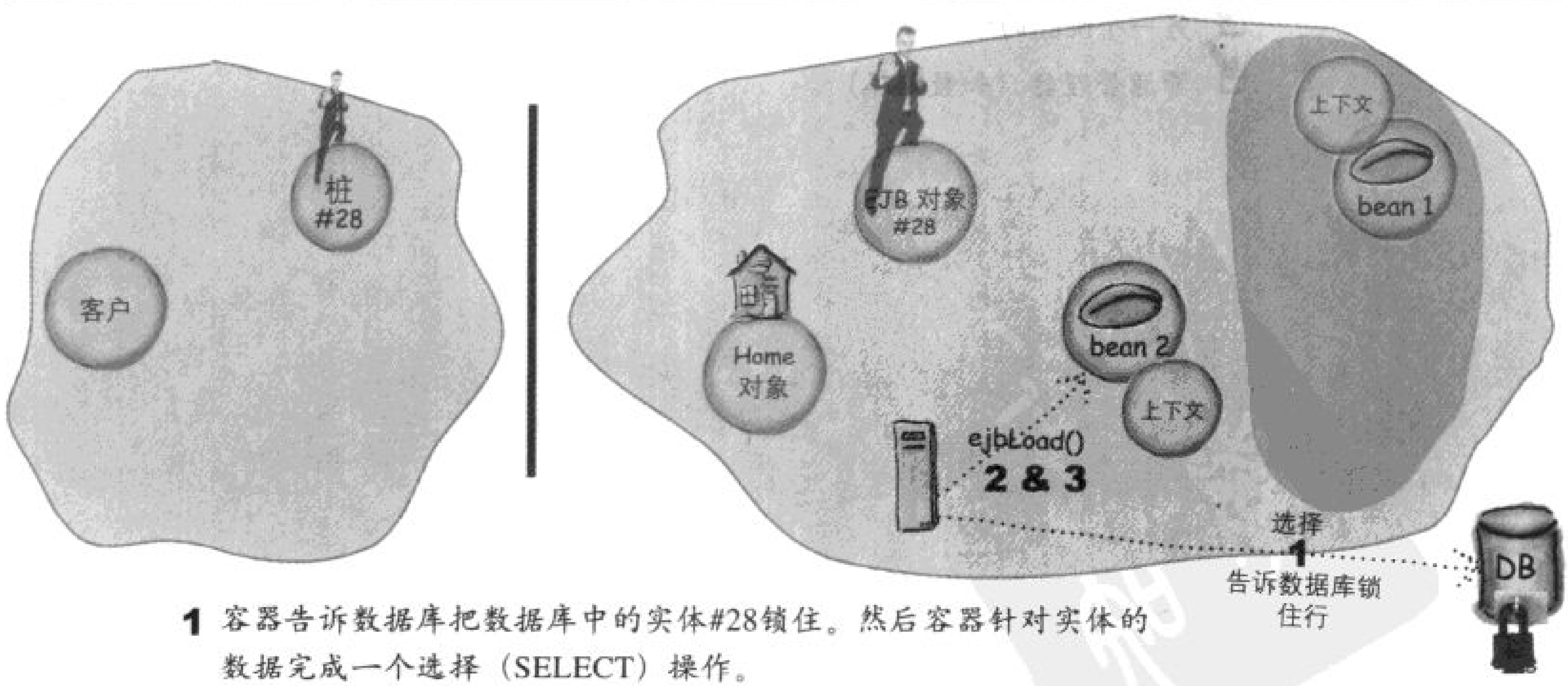

**1** 客户在实体bean#28的桩上调用getAddress()。

**2** 这个调用传递给EJB对象。

**3** EJB对象得到这个调用，着急起来，因为并没有对应#28的实体！

**4** 容器发现这个方法需要一个事务，所以容器开始一个事务，然后从池中取出一个bean，并在这个bean上调用ejbActivate()。

---



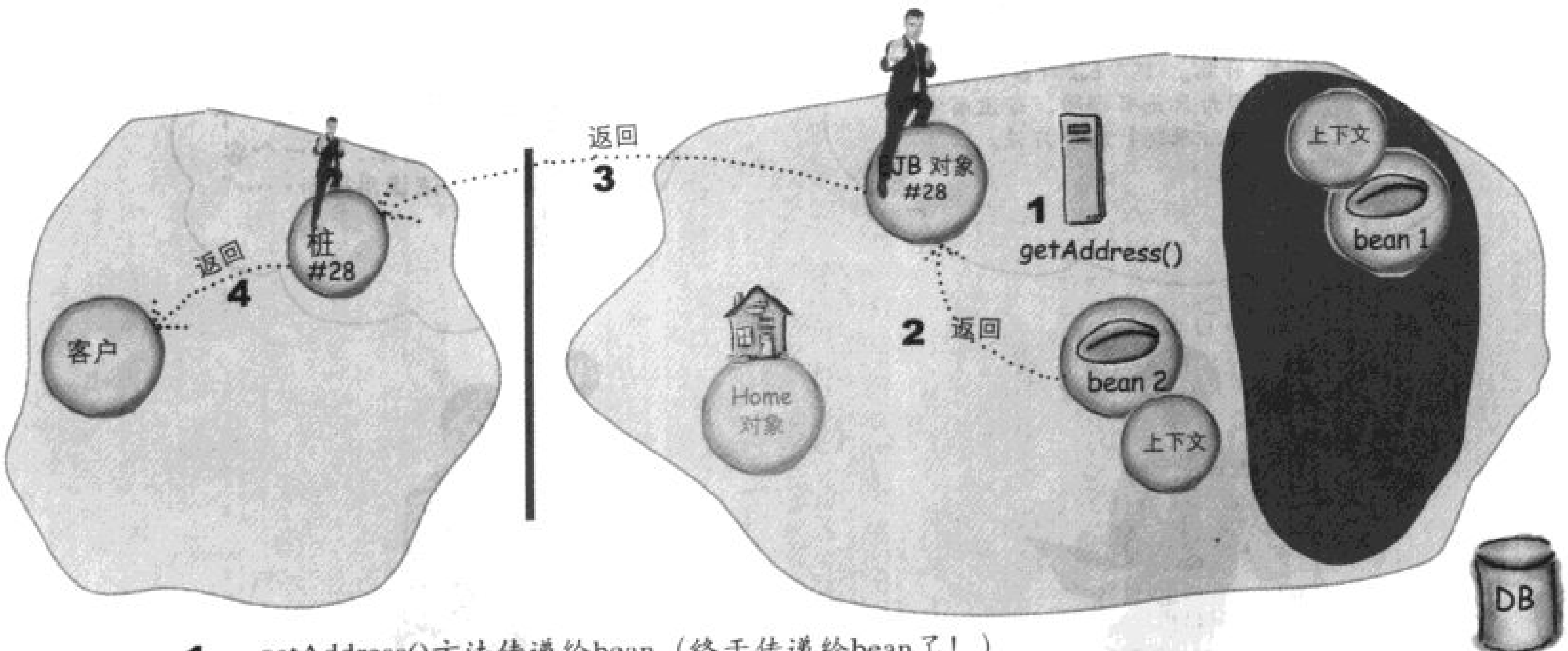

**1** 容器告诉数据库把数据库中的实体#28锁住。然后容器针对实体的数据完成一个选择（SELECT）操作。

**2** 容器用实际的实体数据填充实体bean的持久字段。

**3** 容器在bean上调用ejbLoad()，并告诉bean：“你已经被加载了。”

# 完成一个业务方法（和事务）

场景：bean 完成一个结束事务的业务方法

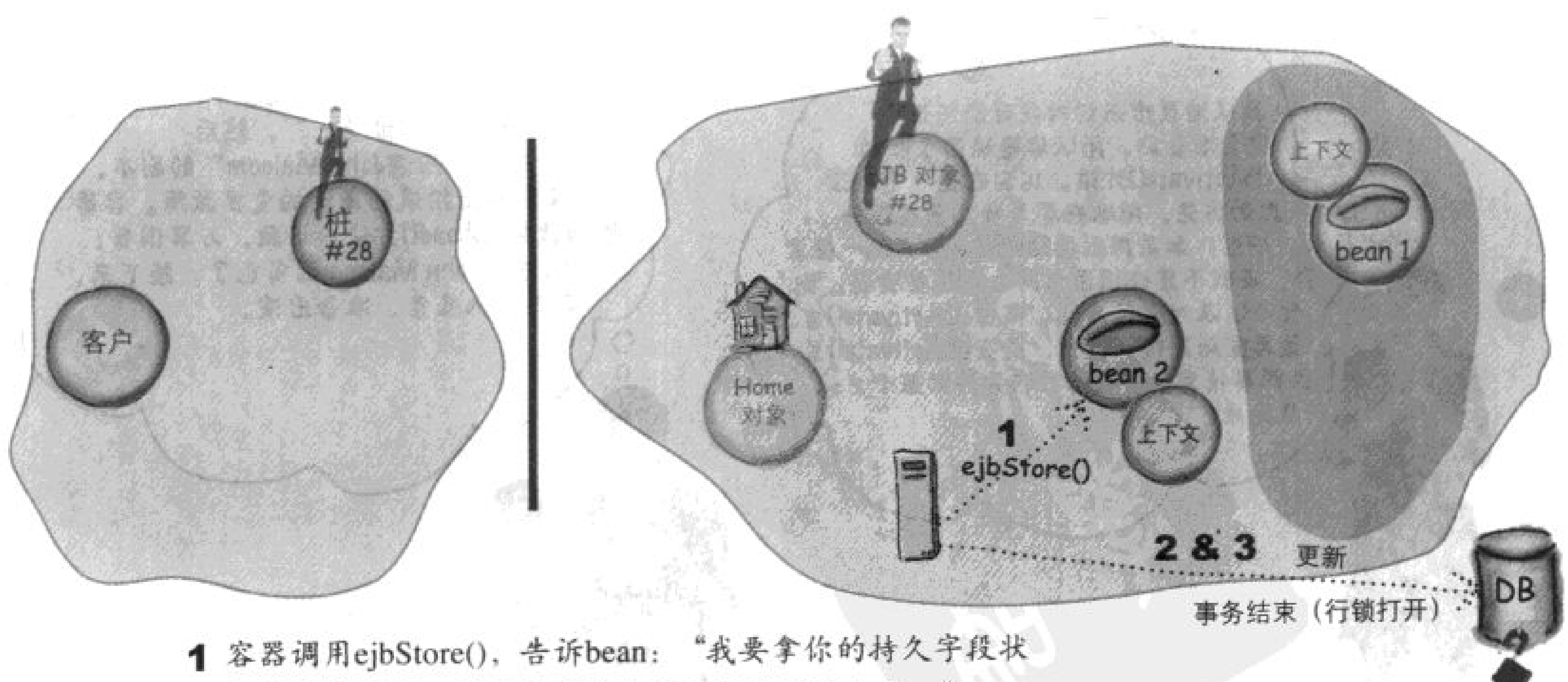



**1** getAddress()方法传递给bean（终于传递给bean了！）。

**2** bean从这个方法返回。

**3/4** EJB对象把返回值发回给客户。

---




**1** 容器调用ejbStore()，告诉bean："我要拿你的持久字段状态更新数据了，所以请确保你的状态已经准备好。"

**2** 数据库中的实际实体得到更新。

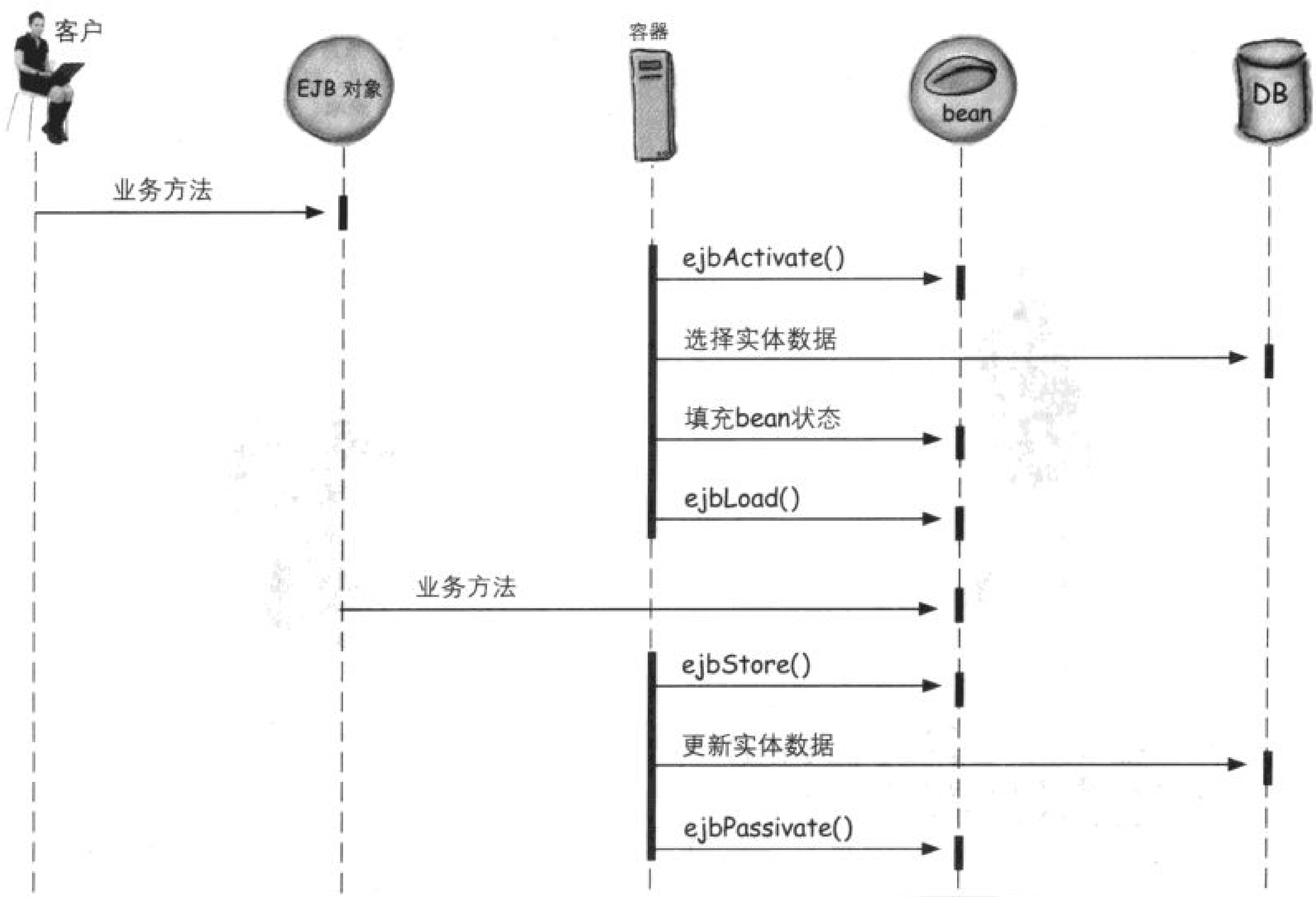
**3** 容器提交事务，告诉数据库把实体的锁打开。

# 当个实体bean——John




5
开演!客户的业务方法调用最终于传给我了（嘿，有没有人告诉我的保镖先休息一下…… 要知道，一直这么精神抖擞可不容易……）。
6
等我以出色表现演完John Malcom时，容器（她有点太严肃了）调用我的ejbStore()方法，来看我有没有修改John的某个内部状态值。这是当然的，我演的这个人物经历了磨难、遭受了苦痛，而且作为一个实体，业务逻辑也有所增长，他当然已经大变样了，可以说变成了另外的一个人。
7
所以我说，“别搞错了……数据在这里”（在他们看来，如果数据库没能了解到人物变化的每一个小细节，就好像到了世界末日一样）。
8
好了，一切结束了，我要回到池里去了。不过，首先，容器要调用我的ejbPassivate()方法，告诉我John Malcom已经演完了，现在可以安全地释放代表他占用的所有资源。我几乎不会用到这个方法，因为这种管理的效率很低下。

## CMP实体bean的激活/钝化




容器总是在ejbActivate()之后调用ejbLoad()。
容器总是在ejbPassivate()之前调用ejbStore()。
容器也可以在其他时间调用ejbStore()和ejbLoad()。

## Sharpen your pencil



如果你要参加考试，一定要听我们的话，现在确实需要停下来，先完成下面这个练习，然后再翻到下一页。好好想想你在这一章学到的有关实体bean生命周期的所有内容，特别是bean什么时候以及如何从池中出来成为一个实体。对于ejbActivate()、ejbPassivate()等4个容器回调，各个容器回调期间可以做哪些bean工作？请尽量做出正确的猜测。

### 使用EntityContext可以：

| ejbActivate() | ejbPassivate() | ejbLoad() | ejbStore() | |
|---|---|---|---|---|
| ❑ | ❑ | ❑ | ❑ | 得到home的一个引用 |
| ❑ | ❑ | ❑ | ❑ | 得到有关客户的安全信息 |
| ❑ | ❑ | ❑ | ❑ | 要求事务回滚 |
| ❑ | ❑ | ❑ | ❑ | 得出事务是否已经置为回滚 |
| ❑ | ❑ | ❑ | ❑ | 得到EJB对象的一个引用 |
| ❑ | ❑ | ❑ | ❑ | 得到主键 |

### 可以访问：

| ejbActivate() | ejbPassivate() | ejbLoad() | ejbStore() | |
|---|---|---|---|---|
| ❑ | ❑ | ❑ | ❑ | 另一个bean的方法 |
| ❑ | ❑ | ❑ | ❑ | 特别的JNDI上下文 |
| ❑ | ❑ | ❑ | ❑ | 资源管理器（如数据库） |



# 激活和加载期间可以做的bean工作

时间 →

### ejbActivate()

使用 EntityContext可以：

- ☑ 得到home的一个引用
- ☑ 得到EJB对象的一个引用
- ☑ 得到主键
- ☐ 得到有关客户的安全信息
- ☐ 要求事务回滚（CMT bean）
- ☐ 查看事务是否已经置为回滚（CMT bean）
- ☐ 得到一个事务引用，并在它之上调用方法（仅适用于BMP bean，所以实体不能使用这个事务引用）

可以访问：

- ☑ 特殊的JNDI环境
- ☐ 另一个bean的方法
- ☐ 资源管理器（如数据库）

容器希望（不，应该说是坚持）你这样：如果你访问一个bean或者一个资源管理器，必须处在一个"有意义的事务上下文中"，所以ejbActivate()还太早，不能做访问bean或资源管理器的事情。不过，一旦达到ejbLoad()，就处在一个事务中了（除非你已经告诉容器不使用事务……这只适用于非常特殊的情况）。

### ejbLoad()

使用 EntityContext可以：

- ☑ 得到home的一个引用
- ☑ 得到EJB对象的一个引用
- ☑ 得到主键
- ☑ 得到有关客户的安全信息
- ☑ 要求事务回滚（CMT bean）
- ☑ 查看事务是否已经置为回滚（CMT bean）
- ☐ 得到一个事务引用，并在它之上调用方法（仅适用于BMP bean，所以实体不能使用这个事务引用）

可以访问：

- ☑ 特殊的JNDI环境
- ☑ 另一个bean的方法
- ☑ 资源管理器（如数据库）

现在可以做实体bean能做的任何事情了……业务方法里能做什么，这里也能做什么。可以把这看作是业务方法的开始。

# 钝化和存储期间可以做的bean工作

时间 →

**ejbPassivate()**

使用 EntityContext可以：

- ☑ 得到home的一个引用
- ☑ 得到EJB对象的一个引用
- ☑ 得到主键
- ☐ 得到有关客户的安全信息
- ☐ 要求事务回滚（CMT bean）
- ☐ 查看事务是否已经置为回滚（CMT bean）
- ☐ 得到一个事务引用，并在它之上调用方法（仅适用于BMP bean，所以实体不能使用这个事务引用）

**可以访问：**

- ☑ 特殊的JNDI环境
- ☐ 另一个bean的方法
- ☐ 资源管理器（如数据库）

容器开始钝化你（bean）的时候，你不再与一个客户或一个事务关联。

就像ejbActivate()一样，容器不再认为你在一个"有意义的事务上下文"中（换句话说，此时你不可能在一个事务中）。所以不能得到一个bean的引用，也不能得到一个资源管理器的引用。容器会说，如果没有一个"有意义的事务上下文"，做这些事情太危险了（请见有关事务的一章）。

**ejbStore()**

使用 EntityContext可以：

- ☑ 得到home的一个引用
- ☑ 得到EJB对象的一个引用
- ☑ 得到主键
- ☑ 得到有关客户的安全信息
- ☑ 要求事务回滚（CMT bean）
- ☑ 查看事务是否已经置为回滚（CMT bean）
- ☐ 得到一个事务引用，并在它之上调用方法（仅适用于BMP bean，所以实体不能使用这个事务引用）

**可以访问：**

- ☑ 特殊的JNDI环境
- ☑ 另一个bean的方法
- ☑ 资源管理器（如数据库）

参考——这一页与有关激活和加载的上一页完全相同（只是我们加的注解不一样）。激活时能做什么，钝化时也能做什么，二者的规则是一样的，另外加载和存储的规则也相同。

# 提交选择：事务提交之后bean到底发生了什么？

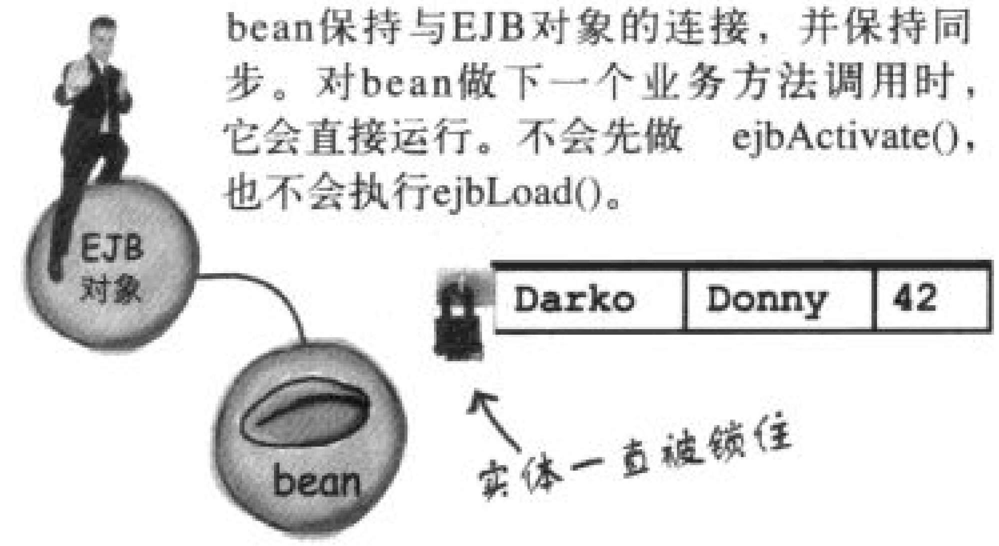
（前面的图只是一种可能，另外还有两种可能）

## ① 提交选择A

bean处于准备状态，与EJB对象关联，并且已经加载了数据，但是不在一个事务中。容器把实体锁住，这样任何人都不能改变bean的状态。客户在这个实体bean的EJB对象上调用一个业务方法时，容器不做ejbActivate()或ejbLoad()，而且假设bean的数据仍与底层持久存储库保持同步。

**A**

bean保持与EJB对象的连接，并保持同步。对bean做下一个业务方法调用时，它会直接运行。不会先做 ejbActivate()，也不会执行ejbLoad()。




## ② 提交选择B

bean保持与EJB对象的关联，已经加载了数据，但是bean的状态被标记为“无效”。换句话说，容器知道这个bean在现在和下一次客户对这个bean调用方法之间可能会过时，所以对这个实体有一个新的业务方法调用时，容器会做一个ejbLoad()（但是不做ejbActivate()）。

**B**

bean保持与EJB对象的连接，但是不同步。对bean做下一个业务方法调用时，会调用ejbLoad()（但是不运行ejbActivate()）。




## ③ 提交选择C

bean被钝化，并放回到池中。下一次客户对这个实体调用一个业务方法时，在这一章前面看到的所有事情都会发生，bean从池中出来，激活，然后加载，最后把业务方法传递给bean。

**C**

bean与EJB对象断开连接，并回到池中。下一次对这个bean做业务方法调用时，容器从池中取出一个bean，然后调用ejbActivate()和ejbLoad()。



## there are no Dumb Questions

**问：** **这些提交选择对我有什么影响？我需要知道我的容器选择的是哪一种吗？我要做选择吗？**

**答：** 根据规范，你的开发商可以自由地使用上述3种选择中的任何一种。你的容器可能允许你选择（或者调整），但是别太指望。那么你需要知道使用的是哪一种选择吗？

当然，可以看出来这三者之间的权衡，如果把实体一直锁在数据库中，那么其他应用都无法使用这个实体的数据。如果EJB是惟一需要这个数据库的应用，这倒还好。另外，对实体做业务方法调用时，这种选择（提交选择A）可以提供最佳的性能，因为这里不做ejbActivate()（bean不会回到池里），更重要的是，不用对数据库做往返通信，也不用调用ejbLoad()!

**问：** **但是在这个bean没有使用数据库中的实体时，我不想让数据库锁住实体的话，选择A不是很糟糕吗？**

**答：** 正是出于这个原因，大多数开发商没有将选择A作为默认选择；只有当部署人员或管理员特别做了这个选择时才允许选择A（这里假设起码要支持这种选择）。

**问：** **我发现，选择A还有一个问题，这个问题甚至更糟糕……这是不是意味着堆会不断增长，最后充斥着对象？比如，每次有人使用某个实体的时候，就会创建一个实体bean实例，它会与EJB对象一直连接，是这样吗？池不是更高效吗？因为这样只要有足够的bean为当前正在执行的方法提供服务就行了。**

**答：** 任何一种容器都不能容忍对象持续不断的一直增长下去。就算是一个确实使用选择A的容器，也可以使用某种算法（比如LRU，即最近最少使用）告诉自己："嗯……很长时间没有调用实体 #908上的方法了，所以我要把这个bean钝化，把它放到池中（而且打开行上的锁）。"不过，采用选择A的话，容器可以告诉自己："这个实体#405很活跃。如果每次都去数据库找它，再完成加载/存储/激活/钝化这个完整的周期，就太费劲了，所以我想把它加载上数据，准备好，就算在事务之间也不让它回到池里去。"

**问：** **噢，老兄，我刚想到这种体系结构的一个很不好的地方……就算是你没有让实体bean一直增加下去，那EJB对象呢？？？它们不是会一直长个没完吗？这样每次客户完成一个查找并得到一个桩时，就会为这个实体创建一个EJB对象，而它只会一直待在那里，直到实体本身被删除，是不是这样？ 在任何给定时刻，堆上都有所有的EJB对象，它们表示的是到那时为止访问过的每一个实体……**

**答：** 哈……现在你发现了体系结构的"概念"视图和"具体"实现之间的差别了。根据规范，你要假设它的工作就是这样的：客户想要一个实体bean时，要么通过一个创建方法，要么通过一个查找方法，容器会建立该实体的一个EJB对象（"给你#24的EJB对象"，"给你#98的EJB对象"，等等）。作为一个bean提供者（不同于容器/服务器提供者），编写程序时就要按这样来（好像就是这样工作一样）。

但是在内部，容器做的可能与概念体系结构视图完全不一样。假设你是一个容器开发人员，你会怎么实现呢？

一种想法可能是把重担压在客户身上，把所有的信息都放在桩里。例如，客户得到一个EJB对象的桩时，可以确保这个桩知道该实体的惟一ID（也就是主键）。但是EJB对象可能很通用，这样当客户在桩上调用一个方法时，这个桩不是与对应该实体的EJB对象连接，而是与一个全盘适用的EJB对象连接，它知道如何确保方法最终在适当的实体上调用。这样一来，客户没有使用一个EJB对象时，这个EJB对象就不会空坐在那里浪费空间。这只是一种想法。

**问：** **这对于桩可能很好，但是如果客户是本地的呢？这样它们就会有EJB对象的一个实际引用，这么一来，EJB不就必须是特定于实体的吗？**

**答：** 不错，是这样的。但是对于本地客户，用正常的Java垃圾回收就行。如果任何客户都没有一个特定EJB对象的引用，这个实例就会被撤销。

## there are no Dumb Questions

**问：这三种提交选择中，你认为哪一种最好用？**

**答：** 嗯……这个问题无法回答，但是如果打赌的话，我们赌选择B。一个"聪明的"选择B知道什么时候留住bean合适，什么时候不合适。不过，不管怎样，避免ejbActivate()总是不错的，这个方法通常都没有用。

**问：但是数据库访问不是会影响很大吗？与栈上的一个方法调用（激活）相比，这个负载不是大得多吗？**

**答：** 首先，你要知道，激活可能不只是涉及栈上的一个方法调用，因为维护池本身的组织就存在开销。不过，你说得对，数据库访问确实是个大问题。但是……容器可以做各种优化，除了这样做还可以选择什么做法呢？惟一的选择（选择A）会让数据库一直锁住，这往往不是你想要的。

**问：但是如果我的EJB应用确实是惟一使用这个数据库的应用呢？**

**答：** 我觉得前面提到过这一点……如果你有绝对把握，认为你的EJB应用就是惟一访问这个数据库的应用，那好，如果你的开发商支持，采用选择A会有更好的性能。当然，把信息缓存在内存中还是很危险的，因为实体是持久的！所以开发商仍然必须实现一些代码，这样当实体的状态有所修改时可以把它们保存到数据库中。所以就算是没有加载，但存储还是有的。

不过，选择A还有一个潜在的问题。例如，你可能有一个支持集群的服务器，而且这个服务器必须确保实体bean的所有实例都要同步，不只是与数据库同步，所有实例彼此之间也要同步。但这是另外一个问题了。在模式和性能一章中会了解更多这方面的内容。

**问：那么，能不能认为关键就是：只要在事务中，就不用这些加载/存储还有激活/钝化调用呢？**

**答：** 对，基本上是这样。

**问：那是不是事务越长越好呢？**

**答：** 你开玩笑吧？这个回答当然是不，不，不。想想并发性吧。有一个很好的经验（但它也有很少的例外，当然所有经验都不是万无一失的），要根据需要决定事务长短，而不要太长。如果有人需要这个实体，而bean越久处在事务中，这就越有可能让别人等待！当然，也可能事务过短，这意味着你做了太多的加载/存储，而本来这是不需要的，所以关键就是：事务的长短要看你的需要，不要太长。

**问：还有一件事……这是不是说，如果容器使用选择A或选择B，可能根本不会在bean上调用ejbPassivate和ejbActivate？这是不是意味着，我的设计不能指望ejbActivate或ejbPassivate？**

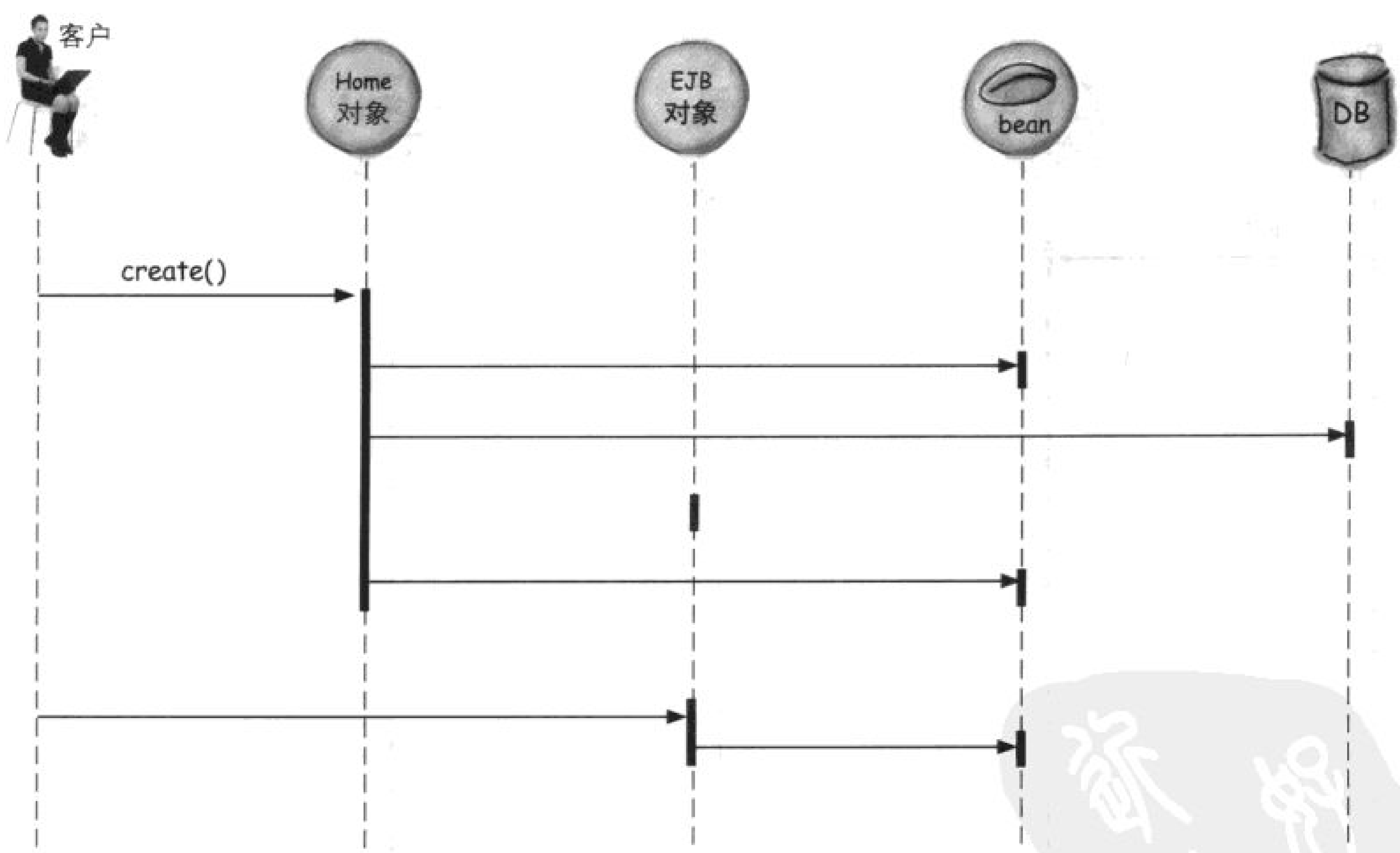
**答：** 对！就是这个意思。这通常是一个很不好的策略。不过，你可以想一下，为什么听上去这个想法好像不错……



练习

## CMP实体bean创建

在对象交互图的5个箭头上填上标签。记住，它们不一定是方法调用。这个图与你在这一章前面看到的图完全相同，只不过没准你的图会更漂亮，因为你可以加上一些奇特的符号，还可以画上有颜色的标记，让它更有趣一些……

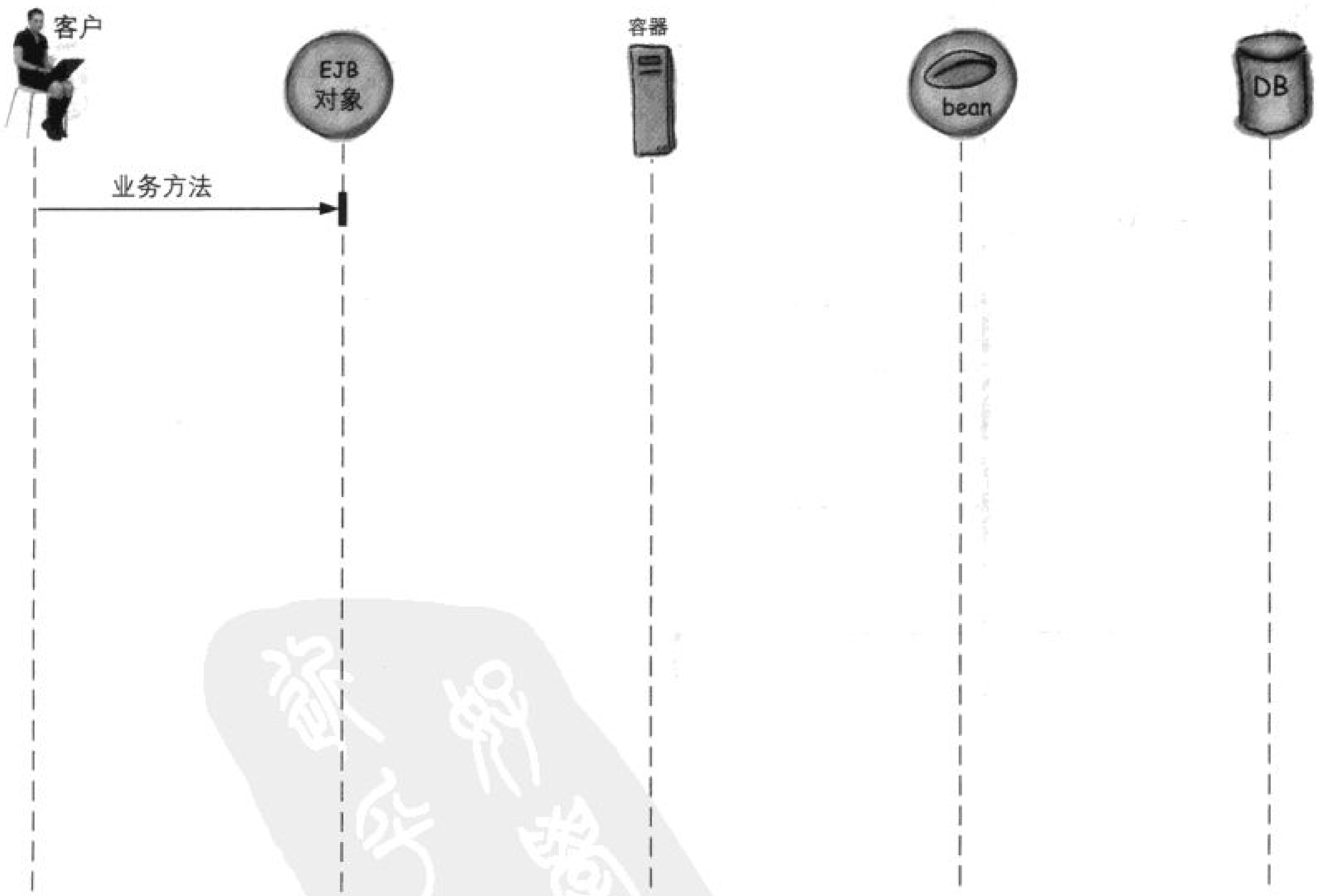





## 实体bean业务方法（带激活和钝化）

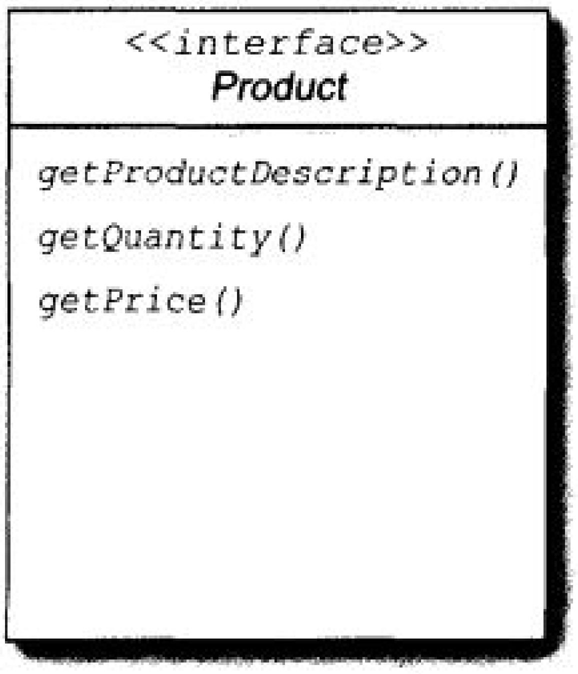
这一次你要自己画箭头了！假设客户调用一个业务方法时，bean正在池里（这也意味着bean不在一个事务中）。画上箭头，并且加上标签，指出业务方法完成过程中发生了哪些动作。说明bean回到池里时会发生什么。

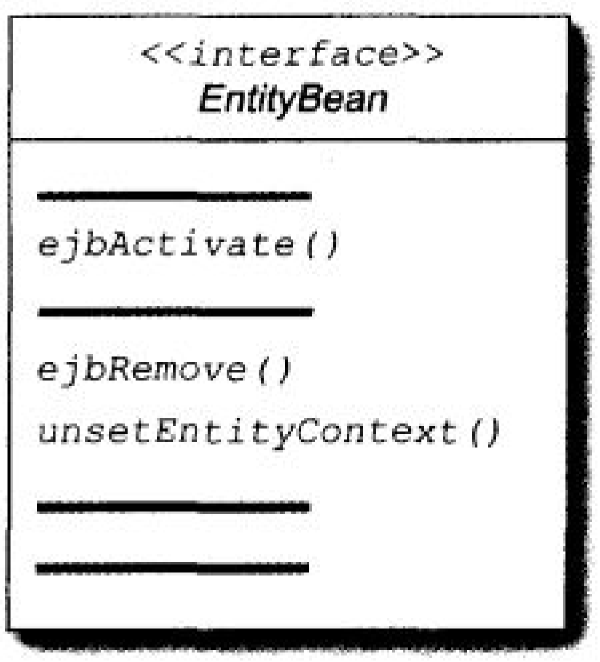
这道题我们自己也做了，多出了8个箭头，不过你的答案可能不一样，可能会多些，也可能会少些，这取决于你想怎么做这道题。不要返回去看前面的几页。你能做出来的。

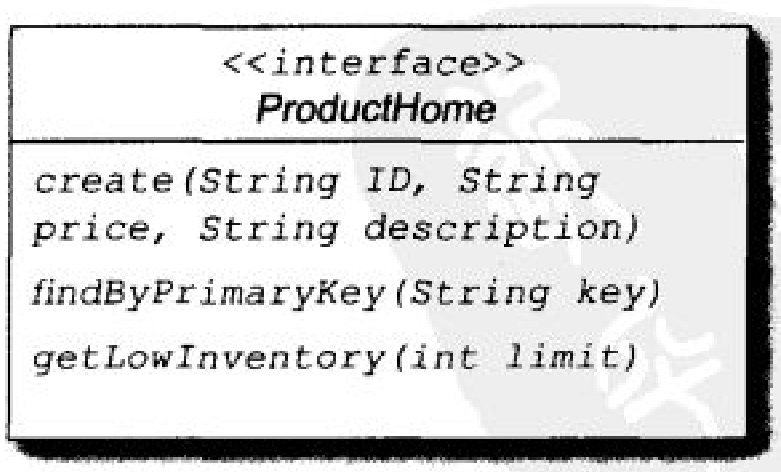


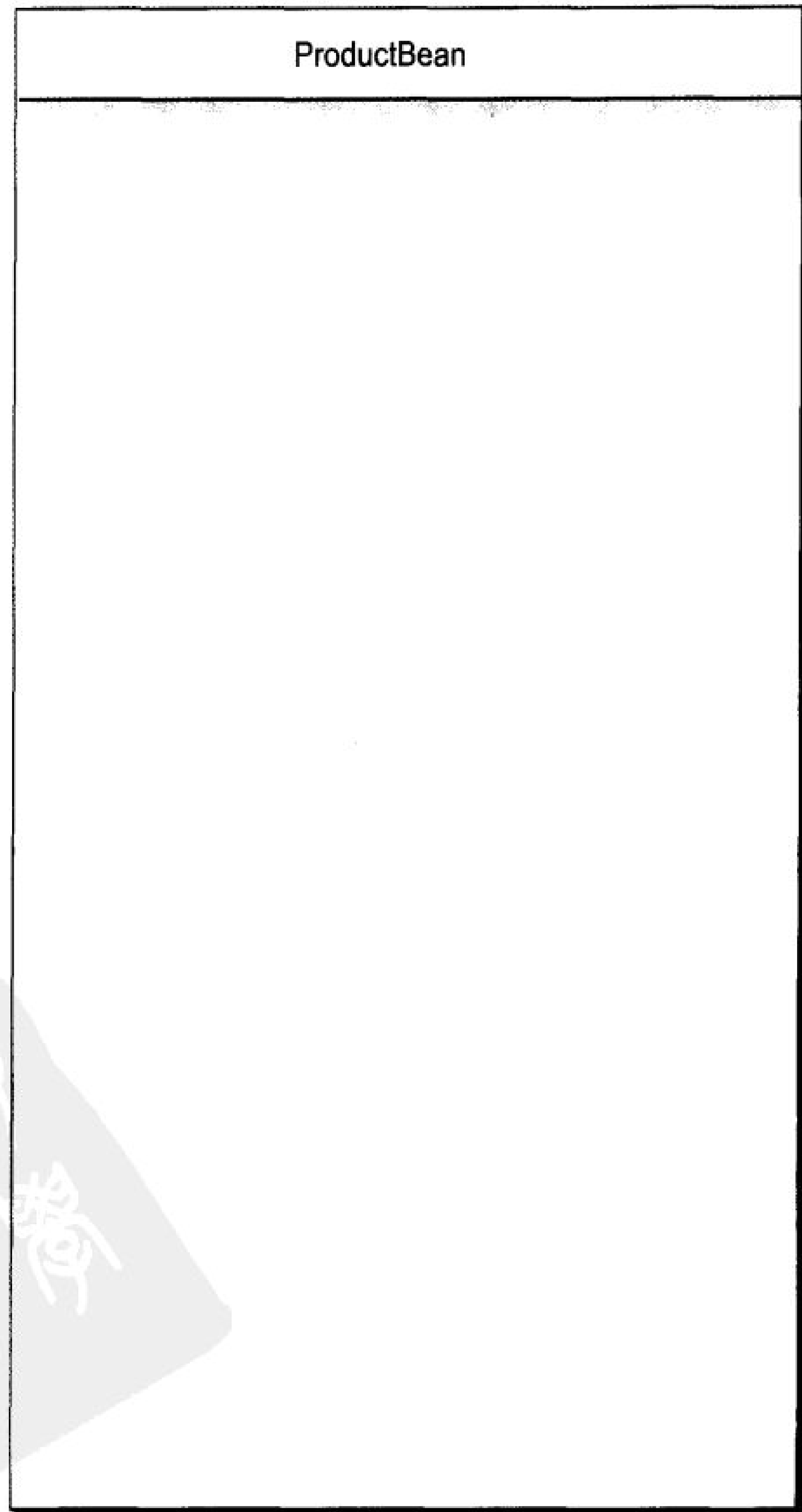



在ProductBean的这个类UML图中填上必须在bean类中编写的方法，这里已经给定了组件接口和home接口。不要忘记EntityBean的容器回调，不过我们只显示了7个容器回调中的3个。余下的要求你自己记住，并写出来。



# COFFEE CRAM
## 模拟测验

**1** 创建一个使用容器托管持久存储的实体bean时，对于一个bean提供者，以下哪些说法是对的？（选出所有正确的答案。）

- ❏ A. 容器托管持久存储字段必须在实体bean类中定义
- ❏ B. 容器托管关系字段必须在实体bean类中定义
- ❏ C. 实现一个一对多关系时，不能使用java.util.List接口
- ❏ D. 容器托管关系字段的存取方法必须在bean的远程组件接口中展现

**2** 对于一个容器托管持久存储实体bean，以下哪些方法是这个实体bean中一个持久字段的合法存取方法？（选出所有正确的答案。）

- ❏ A. `public getCustomerNum();`
- ❏ B. `public void getCustomerNum();`
- ❏ C. `abstract void getCustomerNum();`
- ❏ D. `public abstract int getCustomerNum();`

**3** 对于一个容器托管持久存储实体bean，以下哪些方法是这个实体bean中的合法存取方法？（选出所有正确的答案。）

- ❏ A. `public abstract int GetCustomerNum();`
- ❏ B. `public abstract int getcustomerNum();`
- ❏ C. `public abstract int getCustomerNum();`
- ❏ D. `public abstract int getCustomerNum() { };`

**4** CMP实体bean类有哪些需求？（选出所有正确的答案。）

- ❑ A. 这个类必须定义一个**finalize()**方法
- ❑ B. 源文件必须至少定义一个构造函数
- ❑ C. 这个类必须声明为公共（public）而且抽象（abstract）
- ❑ D. 这个类必须实现（直接或间接）**javax.ejb.EnterpriseBean**接口
- ❑ E. bean抽象持久存储模式的所有获取方法和设置方法都必须是抽象的（abstract）

**5** 以下哪些是CMP bean中ejbCreate方法的合法声明？（选出所有正确的答案。）

- ❑ A. **public void ejbCreateBigCustomer() throws javax.ejb.CreateException**
- ❑ B. **public String ejbCreateAccount() throws javax.ejb.CreateException**
- ❑ C. **static String ejbCreate() throws javax.ejb.CreateException**
- ❑ D. **public int ejbCreate() throws javax.ejb.CreateException**
- ❑ E. **public final String ejbCreate() throws javax.ejb.CreateException**

**6** 以下哪些是CMP bean中一个方法的合法声明？（选出所有正确的答案。）

- ❑ A. **public Account ejbSelectAcct(long x) throws javax.ejb.FinderException**
- ❑ B. **public abstract Acct ejbSelectAcct(long x) throws javax.ejb.FinderException**
- ❑ C. **public Account ejbPostCreate(Acct key) throws javax.ejb.CreateException**
- ❑ D. **public void ejbPostCreate(Acct key) throws javax.ejb.CreateException**
- ❑ E. **public static void ejbPostCreate(Acct key) throws javax.ejb.CreateException**

**7** EntityContext接口的哪些方法可以从setEntityContext方法中调用？（选出所有正确的答案。）

❑ A. **getEJBObject()**

❑ B. **getEJBLocalHome()**

❑ C. **getCallerIdentity()**

❑ D. **getCallerPrincipal()**

❑ E. **setRollbackOnly()**

**8** 可以对一个CMP bean调用哪个方法让它从准备状态迁移到入池状态？（选出所有正确的答案。）

❑ A. **ejbStore()**

❑ B. **ejbCreate()**

❑ C. **ejbSelect()**

❑ D. **ejbRemove()**

❑ E. **ejbPassivate()**

**9** EntityContext接口的哪些方法可以从ejbCreate方法中调用？（选出所有正确的答案。）

❑ A. **getEJBHome()**

❑ B. **getEJBObject()**

❑ C. **getCallerPrincipal()**

❑ D. **getUserTransaction()**

❑ E. **setRollbackOnly()**

**10** 对于处在准备状态的CMP bean，以下哪种说法是正确的？（选出所有正确的答案。）

❑ A. 可以在ejbStore之后直接调用**ejbLoad()**

❑ B. 可以在业务方法之后直接调用**ejbStore()**

❑ C. 可以在ejbStore之后直接调用它的某个业务方法

❑ D. 以上都不对

**11** EntityContext接口中的哪些方法不能在ejbLoad方法中调用？（选出所有正确的答案。）

- ❑ A. `getEJBHome()`
- ❑ B. `getEJBObject()`
- ❑ C. `getCallerPrincipal()`
- ❑ D. `getUserTransaction()`
- ❑ E. `setRollbackOnly()`

**12** CMP bean上调用的哪些方法总是与bean中的一个状态改变相关联？（选出所有正确的答案。）

- ❑ A. `ejbLoad()`
- ❑ B. `ejbFind()`
- ❑ C. `ejbRemove()`
- ❑ D. `ejbActivate()`
- ❑ E. `unsetEntityContext()`

**13** 关于CMP实体bean的主键，以下哪种说法是正确的？（出所有正确的答案。）

- ❑ A. bean的主键类必须提供hashCode以及equals方法的一个合适的实现
- ❑ B. 在部署描述文件中指定主键时，只需声明字段名
- ❑ C. 主键类中的所有字段都必须声明为public
- ❑ D. 主键类中所用的所有字段都必须是容器托管字段

**14** CMP实体bean可以有多少个ejbCreate方法？

- ❑ A. 0个
- ❑ B. 1个
- ❑ C. 0或1个
- ❑ D. 0到多个
- ❑ E. 1到多个

**15** 哪些方法总是作为客户操作的直接响应来调用的？（选出所有正确的答案。）

- ❑ A. **ejbLoad()**
- ❑ B. **ejbCreate()**
- ❑ C. **ejbRemove()**
- ❑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**
- ❑ F. **setEntityContext()**

**16** 调用ejbRemove时容器还可能调用哪些方法？（选出所有正确的答案。）

- ❑ A. **ejbFind()**
- ❑ B. **ejbLoad()**
- ❑ C. **ejbStore()**
- ❑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**

**17** 容器在什么时候必须建立CMP bean的主键？（选出所有正确的答案。）

- ❑ A. 调用**newInstance()**之前
- ❑ B. 调用**setEntityContext()**之前
- ❑ C. 调用 **ejbCreate()**之前
- ❑ D. 调用**ejbPostCreate()**之前

**18** 哪些方法在其调用方法（即导致这个方法得到调用的上层方法）的事务上下文中运行？（选出所有正确的答案。）

- ❑ A. **ejbLoad()**
- ❑ B. **ejbRemove()**
- ❑ C. **ejbSelect()**
- ❑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**
- ❑ F. **setEntityContext()**

# COFFEE CRAM 模拟测验答案

**1** 创建一个使用容器托管持久存储的实体bean时，对于一个bean提供者，以下哪些说法是对的？（选出所有正确的答案。）

❑ A. 容器托管持久存储字段必须在实体bean类中定义

❑ B. 容器托管关系字段必须在实体bean类中定义

✔ C. 实现一个一对多关系时，不能使用java.util.List接口 — 只能用Collection或Set。

❑ D. 容器托管关系字段的存取方法必须在bean的远程组件接口中展现

**2** 对于一个容器托管持久存储实体bean，以下哪些方法是这个实体bean中一个持久字段的合法存取方法？（选出所有正确的答案。）

❑ A. `public getCustomerNum();`

❑ B. `public void getCustomerNum();`

❑ C. `abstract void getCustomerNum();`

✔ D. `public abstract int getCustomerNum();`

— 必须是"public abstract"，而且要返回这个字段的值……毕竟，这是一个获取方法

**3** 对于一个容器托管持久存储实体bean，以下哪些方法是这个实体bean中的合法存取方法？（选出所有正确的答案。）

❑ A. `public abstract int GetCustomerNum();` — 必须遵循Java命名约定

❑ B. `public abstract int getcustomerNum();`

✔ C. `public abstract int getCustomerNum();`

❑ D. `public abstract int getCustomerNum() { };` — 不合法的Java声明

(规范：190)

**4** CMP实体bean类有哪些需求？（选出所有正确的答案。）

- ❑ A. 这个类必须定义一个`finalize()`方法
- ❑ B. 源文件必须至少定义一个构造函数 — 不，只需默认构造函数
- ☑ C. 这个类必须声明为公共（public）而且抽象（abstract）
- ☑ D. 这个类必须实现（直接或间接）`javax.ejb.EnterpriseBean`接口 — EntityBean扩展了EnterpriseBean
- ☑ E. bean抽象持久存储模式的所有获取方法和设置方法都必须是抽象的

(规范：192)

**5** 以下哪些是CMP bean中ejbCreate方法的合法声明？（选出所有正确的答案。）

- ❑ A. `public void ejbCreateBigCustomer() throws javax.ejb.CreateException`
- ☑ B. `public String ejbCreateAccount() throws javax.ejb.CreateException`
- ❑ C. `static String ejbCreate() throws javax.ejb.CreateException`
- ❑ D. `public int ejbCreate() throws javax.ejb.CreateException`
- ❑ E. `public final String ejbCreate() throws javax.ejb.CreateException`

— 不能是
— static
— void
— final
必须返回主键的类型

(规范：192～193)

**6** 以下哪些是CMP bean中一个方法的合法声明？（选出所有正确的答案。）

- ❑ A. `public Account ejbSelectAcct(long x) throws javax.ejb.FinderException`
- ☑ B. `public abstract Acct ejbSelectAcct(long x) throws javax.ejb.FinderException`
- ❑ C. `public Account ejbPostCreate(Acct key) throws javax.ejb.CreateException`
- ☑ D. `public void ejbPostCreate(Acct key) throws javax.ejb.CreateException`
- ❑ E. `public static void ejbPostCreate(Acct key) throws javax.ejb.CreateException`

— 必须是：
abstract
void
— 不能是
static

**7** EntityContext接口的哪些方法可以从setEntityContext方法中调用？（选出所有正确的答案。）

（规范：179）

- ❑ A. **getEJBObject()**
- ☑ B. **getEJBLocalHome()**
- ❑ C. **getCallerIdentity()**
- ❑ D. **getCallerPrincipal()**
- ❑ E. **setRollbackOnly()**

— 太早了，还不能调用最后这3个方法。此时还没有一个调用者事务。setEntityContext只是意味着bean要到池中

（规范：168～69）

**8** 可以对一个CMP bean调用哪个方法让它从准备状态迁移到入池状态？（选出所有正确的答案。）

- ❑ A. **ejbStore()**
- ❑ B. **ejbCreate()**
- ❑ C. **ejbSelect()**
- ☑ D. **ejbRemove()**
- ☑ E. **ejbPassivate()**

— 存储方法可以在任何时刻调用，调用这个方法不会让bean的状态迁移

— 如果有ejbRemove()，bean就不会得到ejbPassivate()

（规范：179）

**9** EntityContext接口的哪些方法可以从ejbCreate方法中调用？（选出所有正确的答案。）

- ☑ A. **getEJBHome()**
- ❑ B. **getEJBObject()**
- ☑ C. **getCallerPrincipal()**
- ❑ D. **getUserTransaction()**
- ☑ E. **setRollbackOnly()**

— 实体bean不可能调用这个方法，因为实体bean必须使用CMT

（规范：169）

**10** 对于处在准备状态的CMP bean，以下哪种说法是正确的？（选出所有正确的答案。）

- ☑ A. 可以在ejbStore之后直接调用**ejbLoad()**
- ☑ B. 可以在业务方法之后直接调用**ejbStore()**
- ☑ C. 可以在ejbStore之后直接调用它的某个业务方法
- ❑ D. 以上都不对

— 这里的重点是：加载、存储和业务方法可以由容器按任何顺序调用

**11** EntityContext接口中的哪些方法不能在ejbLoad方法中调用？（选出所有正确的答案。） （规范：180）

- ❑ A. `getEJBHome()`
- ❑ B. `getEJBObject()`
- ❑ C. `getCallerPrincipal()`
- ☑ D. `getUserTransaction()` — 仅用于BMT
- ❑ E. `setRollbackOnly()`

**12** CMP bean上调用的哪些方法总是与bean中的一个状态改变相关联？（选出所有正确的答案。） （规范：168～169）

- ❑ A. `ejbLoad()` — 不一定！
- ❑ B. `ejbFind()`
- ☑ C. `ejbRemove()` — bean回到池中
- ☑ D. `ejbActivate()` — bean离开池
- ☑ E. `unsetEntityContext()` — bean从池中出来后"死掉"（删除）

**13** 关于CMP实体bean的主键，以下哪种说法是正确的？（选出所有正确的答案。） （规范：203，275）

- ☑ A. bean的主键类必须提供hashCode以及equals方法的一个合适的实现
- ❑ B. 在部署描述文件中指定主键时，只需声明字段名 — 不对，还需要类型
- ☑ C. 主键类中的所有字段都必须声明为public
- ☑ D. 主键类中所用的所有字段都必须是容器托管字段

**14** CMP实体bean可以有多少个ejbCreate方法？ （规范：171）

- ❑ A. 0个
- ❑ B. 1个
- ❑ C. 0或1个
- ☑ D. 0到多个 不用让客户创建bean
- ❑ E. 1到多个

**15** 哪些方法总是作为客户操作的直接响应来调用的？（选出所有正确的答案。）

（规范：171～172）

- ❑ A. **ejbLoad()**
- ☑ B. **ejbCreate()**
- ☑ C. **ejbRemove()**
- ❑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**
- ❑ F. **setEntityContext()**

容器想调用加载方法、钝化方法和setEntityContext时，可以随时调用

**16** 调用ejbRemove时容器还可能调用哪些方法？（选出所有正确的答案。）

（规范：176）

- ❑ A. **ejbFind()**
- ❑ B. **ejbLoad()**
- ☑ C. **ejbStore()**
- ❑ D. **ejbActivate()**
- ☑ E. **ejbPassivate()**

bean在调用删除方法之前必须准备好，而且要已经加载——可能还要考虑级联删除

**17** 容器在什么时候必须建立CMP bean的主键？（选出所有正确的答案。）

（规范：175）

- ❑ A. 调用**newInstance()**之前
- ❑ B. 调用**setEntityContext()**之前
- ❑ C. 调用**ejbCreate()**之前
- ☑ D. 调用**ejbPostCreate()**之前

容器使用你在ejbCreate中设置的值，这样在ejbPostCreate中bean就可以使用它的键了

**18** 哪些方法在其调用方法（即导致这个方法得到调用的上层方法）的事务上下文中运行？（选出所有正确的答案。）

（规范：174～178）

- ☑ A. **ejbLoad()**
- ☑ B. **ejbRemove()**
- ☑ C. **ejbSelect()**
- ❑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**
- ❑ F. **setEntityContext()**

# 7 实体关系

# bean之间的关系




**实体bean需要关系。**一个订单需要一个顾客。一件商品需要一个订单。一个订单需要多件商品。一部电影需要一个导演。一个导演需要多部电影。一个电影需要多个演员。演员需要天赋……（这就扯远了）。实体bean可以有容器托管关系（CMR），由容器负责几乎所有一切。要拍个新电影并指定一名导演吗？在这个导演的电影作品集中会自动增加一部电影。生产了一个新商品吗？如果这个商品与一个订单关联，你要求顾客给你出示他的订单时，他的订单就会显示出这个新商品。还有最好的一点，这就是可以使用EJB-QL来编写可移植查询（这里的可移植可以理解为：独立于开发商/数据库的）。

# 实体关系：CMP和CMR

## 标准大纲：

6.2 关于CMP实体bean的持久存储关系、删除协议，以及抽象模式类型，区别哪些说法或示例是正确的，哪些不正确。

6.3 关于CMP bean中关系指派更新的规则和语义，区别哪些说法或示例是正确的，哪些不正确。

6.4 给定以下标记：<ejb-name>、<abstract-schema-name>、<ejb-relationship-role>、<cmr-field>、<cmr-field-type>和<relationship-role-source>，将各个标记的名字与其用途或功能描述配对。

6.5 对于一个CMP bean，区分哪些是实现正确的部署描述文件元素（包括容器托管关系）。

9.1 区分一个EJB-QL查询（包括SELECT、FROM和WHERE子句）的语法哪些是正确的，哪些不正确。

9.2 关于EJB-QL的用途和使用，区别哪些说法或示例是正确的，哪些不正确。

9.3 区分哪些条件表达式、 between表达式、like表达式和比较表达式是正确的，哪些不正确。

## 具体含义：

你必须知道容器通过容器托管关系（CMR）字段来管理两个实体bean之间的关系，而且必须在部署描述文件中使用<relationships>部分描述这些CMR字段。

必须理解关系中多重性的影响，还要知道容器很关心关系到底是一对一、一对多，还是多对多。你要知道，容器就是在指派时使用多重性来维护数据库的引用完整性的。例如，如果每个订单只有一个顾客，而且如果你把一个订单指派给了一个顾客，那么这个订单就不能存在于任何其他顾客的订单集中。如果再把这个订单指派给另外一个顾客，订单就会从第一个顾客的订单集转移到第二个顾客的订单集中。不过，如果一个顾客原来有3个订单，把这个顾客指派到另一个订单时，顾客引用只会复制，而不是移动，这样原来3个已有的订单所拥有的顾客引用并不受影响。

必须知道EJB-QL查询是在部署描述文件中定义的，而且只用于查找方法和选择方法。你应该知道EJB-QL的基本语法，其中FROM和SELECT子句是必要的，而WHERE子句可选。

你应该知道，如果使用路径导航来做一些事情，比如说得到一个电影的片名，可以称之为m.title（这里假设m声明为Movie的抽象模式类型，而title是MovieBean的一个CMP字段）。但是不能说d.movies.title，这里d是一个Director（导演），movies是一个CMR字段，这是一个电影Collection。你的本意是想得到一个导演执导的所有电影的片名，为此，必须在一个FROM子句中使用IN操作符。后面就会介绍这些内容。尽管听上去好像很难，但实际上远没有想像中复杂。

# 为电影数据库建立bean

假设你有一个电影应用（这可不是面向大众的电影网站，别人不必为此紧张，这个应用只是供你使用），可以使用这个应用来查找电影。

一旦得到了一个电影，可以用它来做一个电影预告片。

可以做各种搜索，比如说找到所有科幻电影，或者找到某个导演执导的所有动作片。

我们可以为电影（Movie）、预告片（Trailer）和导演（Director）建立bean。但是它们如何映射到数据库？另外，它们彼此之间又如何关联？




与Director表关联的一个外键

Movie表

| MovieID | Title | Genre | DirectorID | Year |
|---|---|---|---|---|
| 12 | Crouching Pixels, Hidden Mouse | Action | 42 | 2000 |
| 5 | The Fifth Array Element | Action | 27 | 2003 |
| 22 | The Return of the Bean Queen | Fantasy | 27 | 2001 |
| 11 | Lord of the Loops | Sci-fi | 42 | 2002 |

Director表

| DirectorID | OscarWinner | Degrees | Name |
|---|---|---|---|
| 27 | TRUE | 3 | Jim Yingst |
| 42 | FALSE | 6 | Jessica Sant |
| 56 | FALSE | 24 | Skyler Safford |
| 17 | FALSE | 5 | John Dawson |

Director没有对他所执导电影的引用。

Trailer表

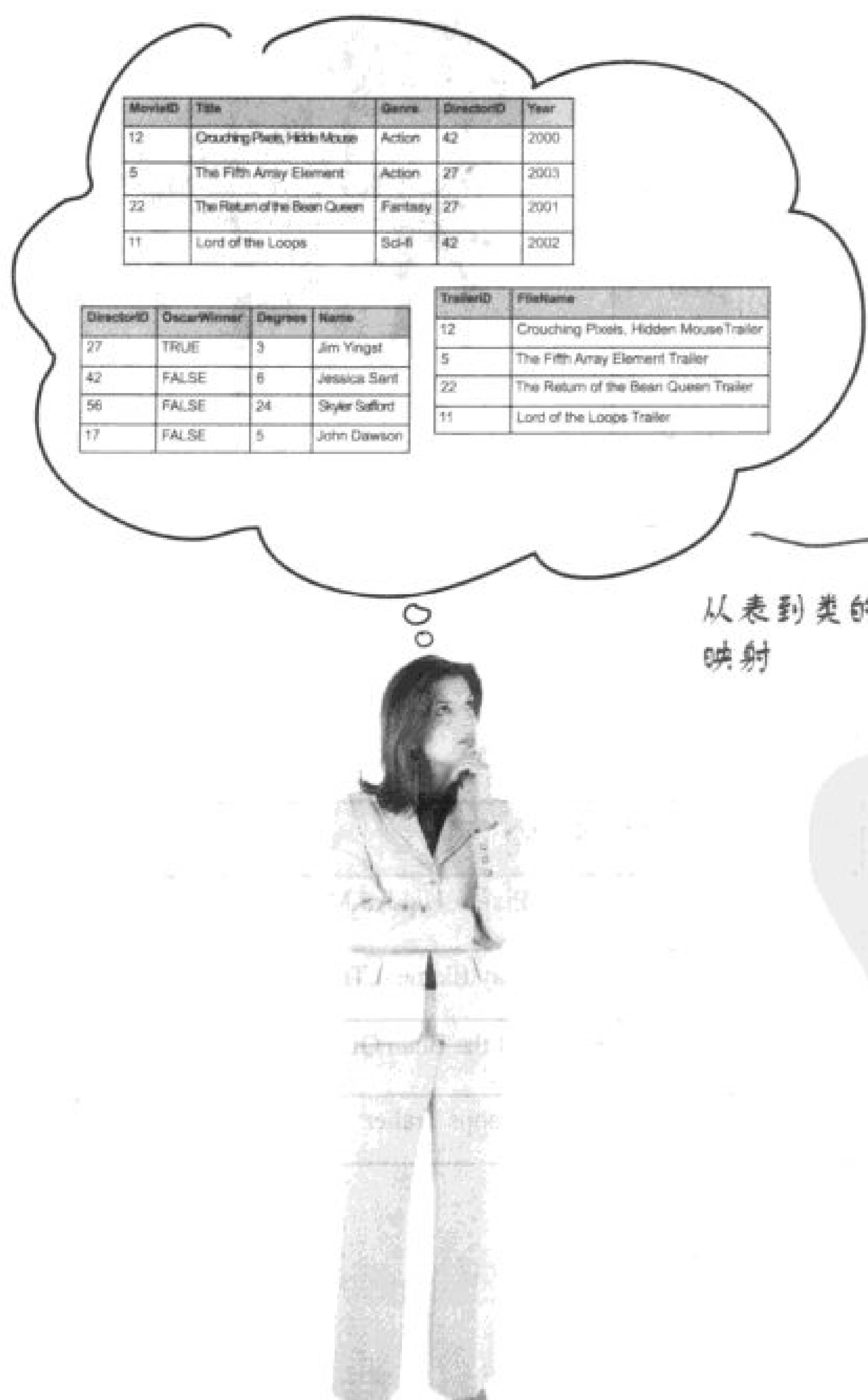
| TrailerID | FileName |
|---|---|
| 12 | Crouching Pixels, Hidden Mouse Trailer |
| 5 | The Fifth Array Element Trailer |
| 22 | The Return of the Bean Queen Trailer |
| 11 | Lord of the Loops Trailer |

预告片的主键总是与MovieID匹配

# 但是我们不想按表来思考，我们想用类的思路考虑

我们知道，可以把表映射到bean类，这没有问题。表中的所有列都可以变成bean类中的持久字段，由抽象的获取方法和设置方法表示，这些我们很在行。但是……如果是一个类，而不是一个表，就不能这样设计了。例如，可能要把电影类做得更加友好，更加有用，这样客户完全可以只使用电影bean，而不必把这3个bean的引用都拿到。

但是这看上去对客户不太友好。看看客户必须做些什么……他要得到所有这3个表的引用！如果客户有一个电影的引用，他们必须从这个电影得到directorID，然后再使用这个directorID得到导演的一个引用。

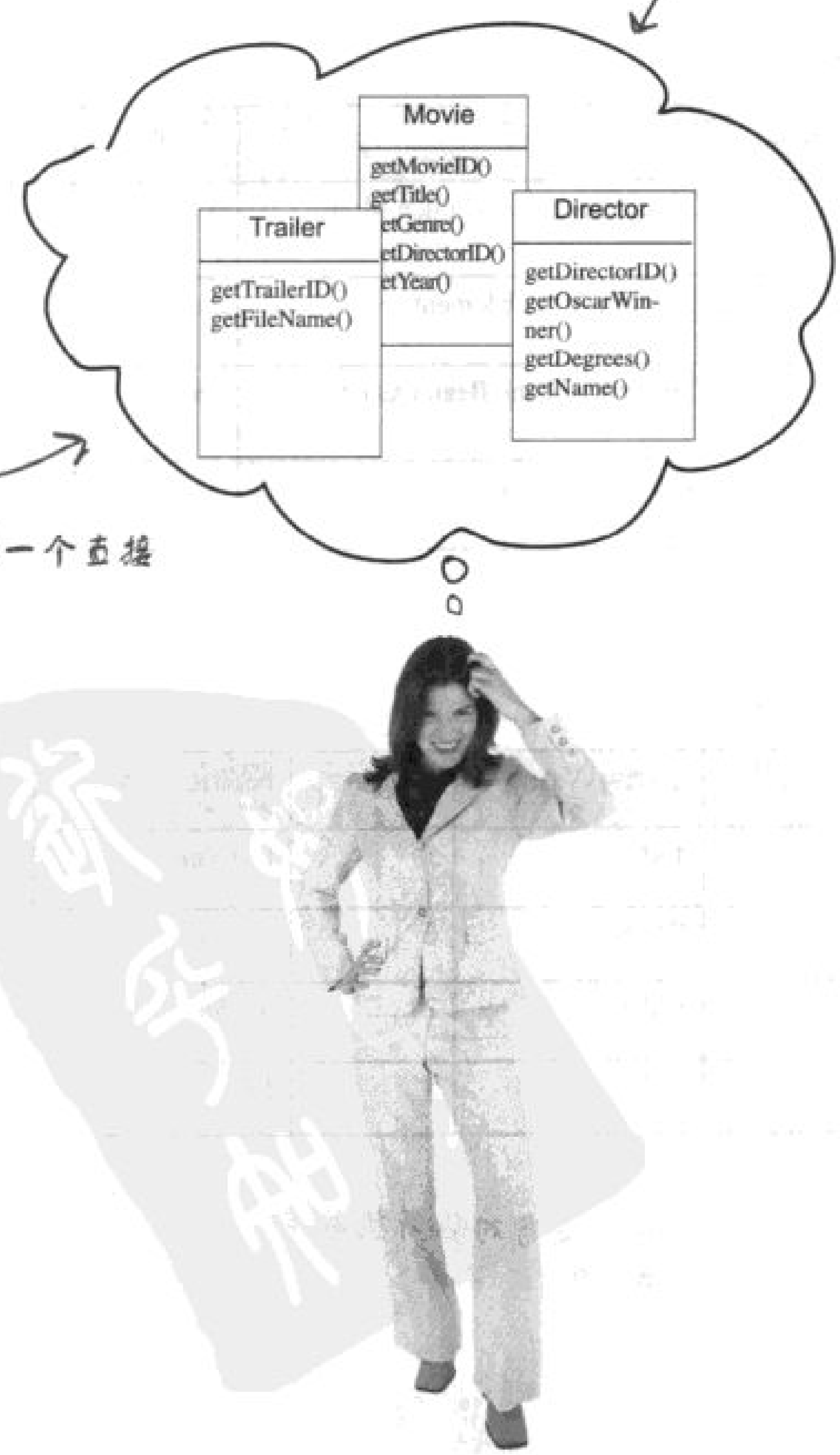


| MovieID | Title | Genre | DirectorID | Year |
|---|---|---|---|---|
| 12 | Crouching Pixels, Hidde Mouse | Action | 42 | 2000 |
| 5 | The Fifth Array Element | Action | 27 | 2003 |
| 22 | The Return of the Bean Queen | Fantasy | 27 | 2001 |
| 11 | Lord of the Loops | Sci-fi | 42 | 2002 |

| DirectorID | OscarWinner | Degrees | Name |
|---|---|---|---|
| 27 | TRUE | 3 | Jim Yingst |
| 42 | FALSE | 6 | Jessica Sant |
| 56 | FALSE | 24 | Skyler Safford |
| 17 | FALSE | 5 | John Dawson |

| TrailerID | FileName |
|---|---|
| 12 | Crouching Pixels, Hidden MouseTrailer |
| 5 | The Fifth Array Element Trailer |
| 22 | The Return of the Bean Queen Trailer |
| 11 | Lord of the Loops Trailer |

从表到类的一个直接映射

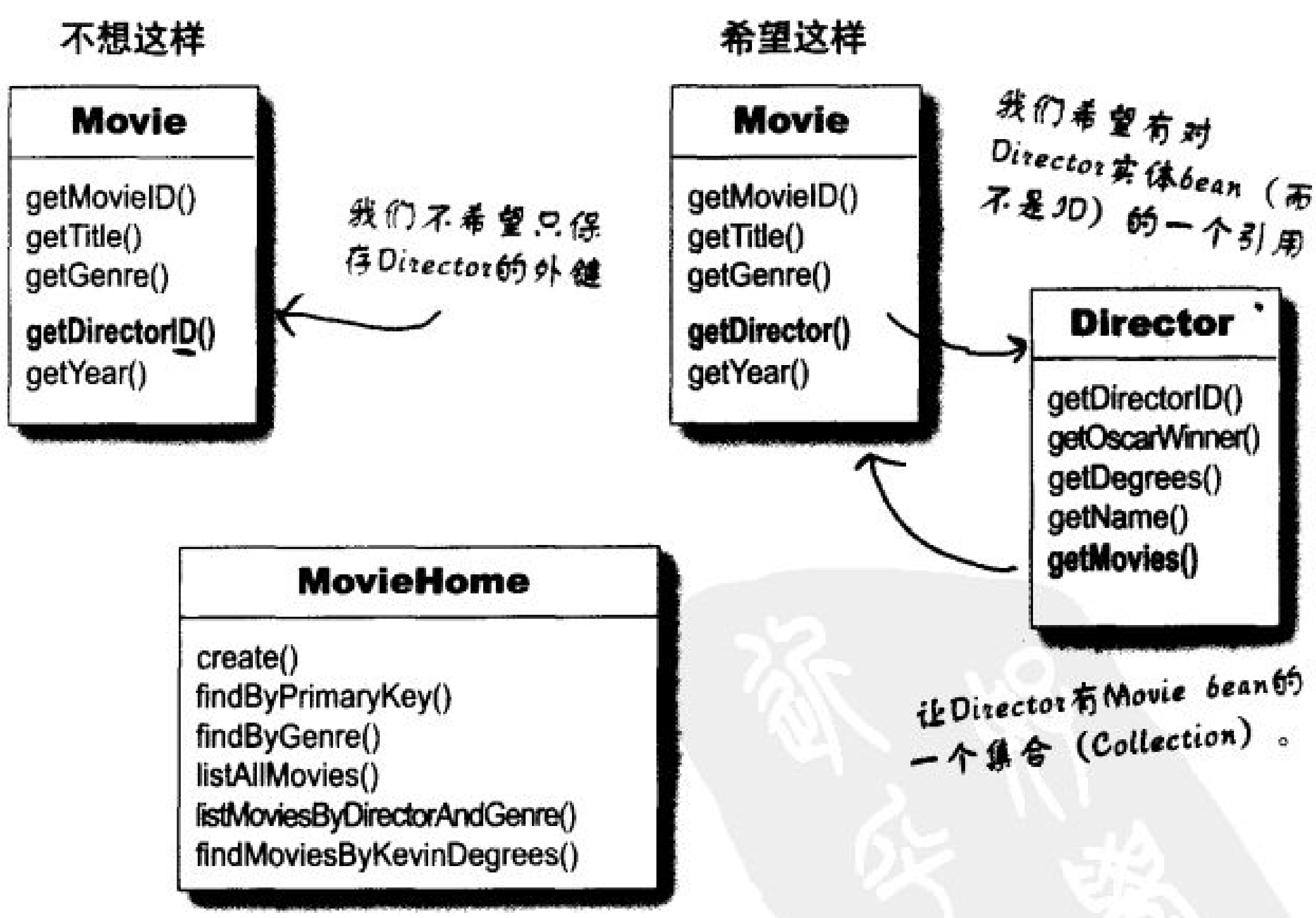



如果有办法使用容器托管持久存储但
同时类不必完全对应数据库表，那该多好
呀，这不只是个梦吧？这样一来，客户就
可以得到一个电影，向这个电影要一个实际的导
演引用，而不只是导演的外键……嗯，要能这样
就太棒了，不过，没准这只是异想天开……

# 我们需要Movie bean和Director bean之间的关系

我们希望Movie bean有相应Director bean的一个引用，各种查找都针对Movie bean完成，而且查询还可以使用Director bean的数据。

换句话说，我们希望客户（以及开发人员）能够更轻松，能用一种更自然的方式思考，而不是采用设计数据库模式的思路，这种方式只是能有利于数据库。有谁想这样呢？要记住，这是一个面向对象世界，有一些关系数据库，尽管这些数据库对你的业务很重要，但是我们只是想使用数据库，而不想像数据库那样思考（如果你很幸运，使用的是一个面向对象数据库，而且假设你的数据库能以某种方式很好地满足你的要求，那你可能会对我们介绍的这一部分不以为然，暗自窃喜不用像我们这样绞尽脑汁地从关系世界转向面向对象世界）。




为了让这个例子更加简洁，我们后面不打算使用Trailer表和Trailer bean（不过在代码中会看到它）。不过，就算是增加Trailer bean，也不会对应用增加更多的复杂性。如果可以建立一个bean-to-bean关系，那同样也可以建立其他的bean-to-bean关系。

# 为什么Director要作为一个bean？为什么不能仅作为数据？

为什么Movie bean不是简单地访问数据库，也就是使用Director的外键（存储为Movie bean中的一个持久字段）来得到Director的数据呢？不过可以这样想，如果把导演也建立为一个bean，就只需从对象的角度来考虑问题（部署时除外，在部署时确实必须从bean映射到数据库表）。而且这样一来，你可以得到容器托管持久存储和同步的所有好处。假设有人调用了一个设置方法来修改导演的奥斯卡得奖情况。如果由你来管理，就必须得到一个JDBC连接，而且你要自己完成与数据库的同步，当然还必须知道什么时间同步等等一系列问题。但是，如果导演也是一个bean，那你只需调用setWinner()方法，剩下的就不用你管了。

哦，对了，还有一点很棒。例如，你可能决定把某个导演从数据库里删除，因为你觉得他的电影实在让人看不下去。显然，删除这个导演没有任何问题，但是他的那些电影会怎么样呢？数据库里怎么能有没有导演的电影呢？如果建立这个数据库时要求导演列不允许为null值该怎么办？也就是说，如果电影没有导演，就无法存在于数据库中，这该怎么办？

这个问题可以轻松地解决，只需用某种方法来建立Movie-to-Director bean关系，这样当你从数据库中删除一个导演时，他的所有电影也会自动删除！为此，只需要在部署描述文件里设置一个简单的标记，就这么简单！如果有人调用了一个导演bean的remove()，那么有这个导演引用的所有电影实体bean上也会自动地调用remove()（也就是说，如果电影实体bean中director持久字段的值就是这个导演引用，那这些电影bean也会自动删除）。

如果Director和Movie都是实体bean，你就不用担心Movie相关数据的同步：如果有人使用了Movie bean来修改Director的某个数据，所有这一切都会由容器负责处理。

你甚至可以设置级联删除……这样一来，如果删除了一个Director，那么这个Director的所有电影都会自动删除！

# 关系和多重性

## Ⓐ 一对一



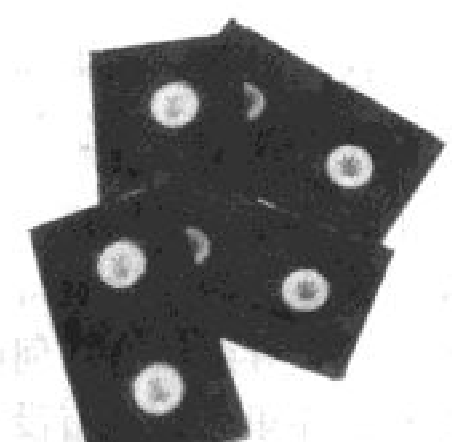


每个 Movie —有→ 一个 Trailer

一个 Movie ←有— 每个 Trailer

## Ⓑ 一对多





一个 Director —有→ 多个 Movie

一个 Director ←有— 每个 Movie

## ③ 多对多



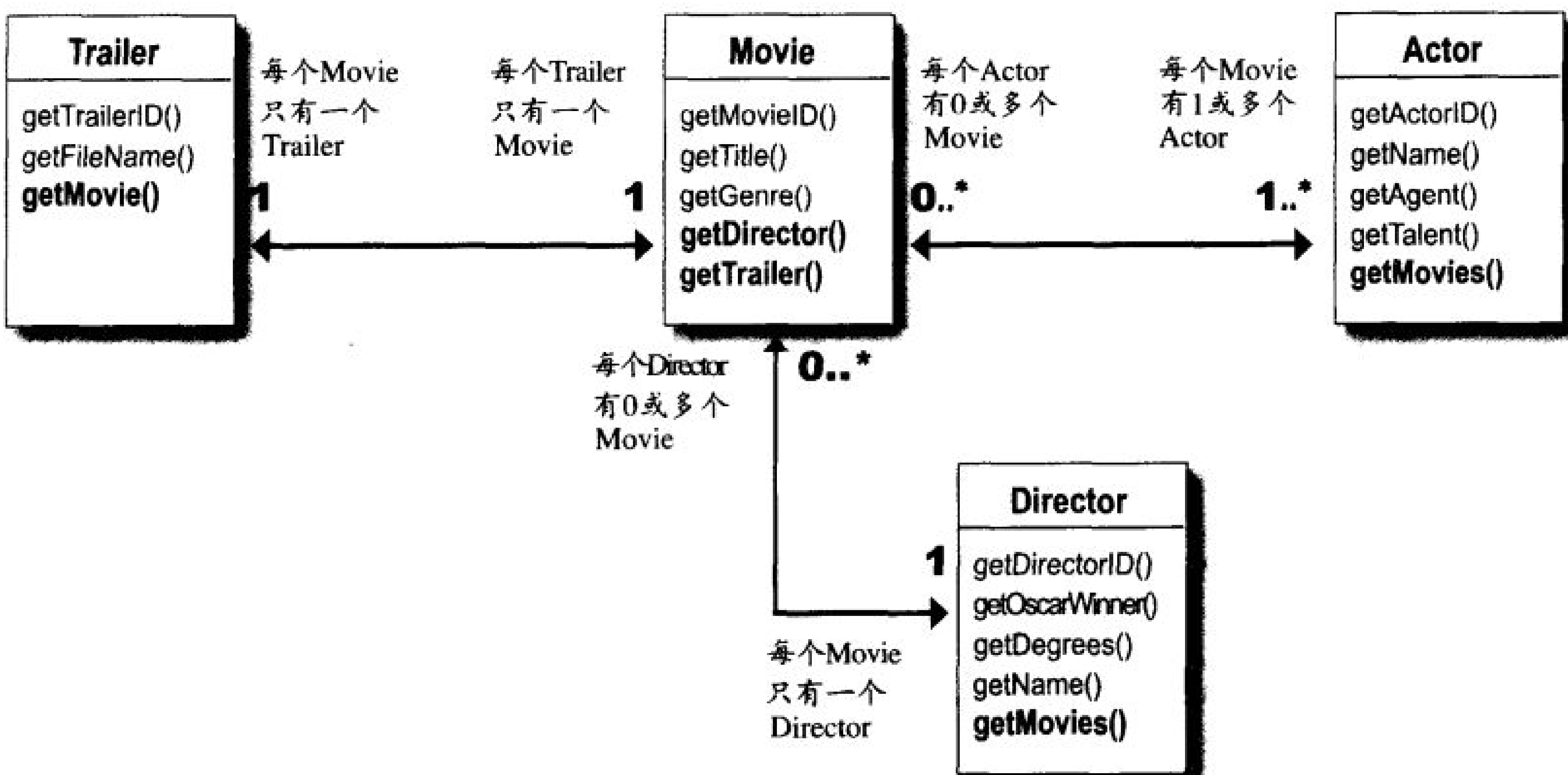


一个 Actor —出演→ 多个 Movie

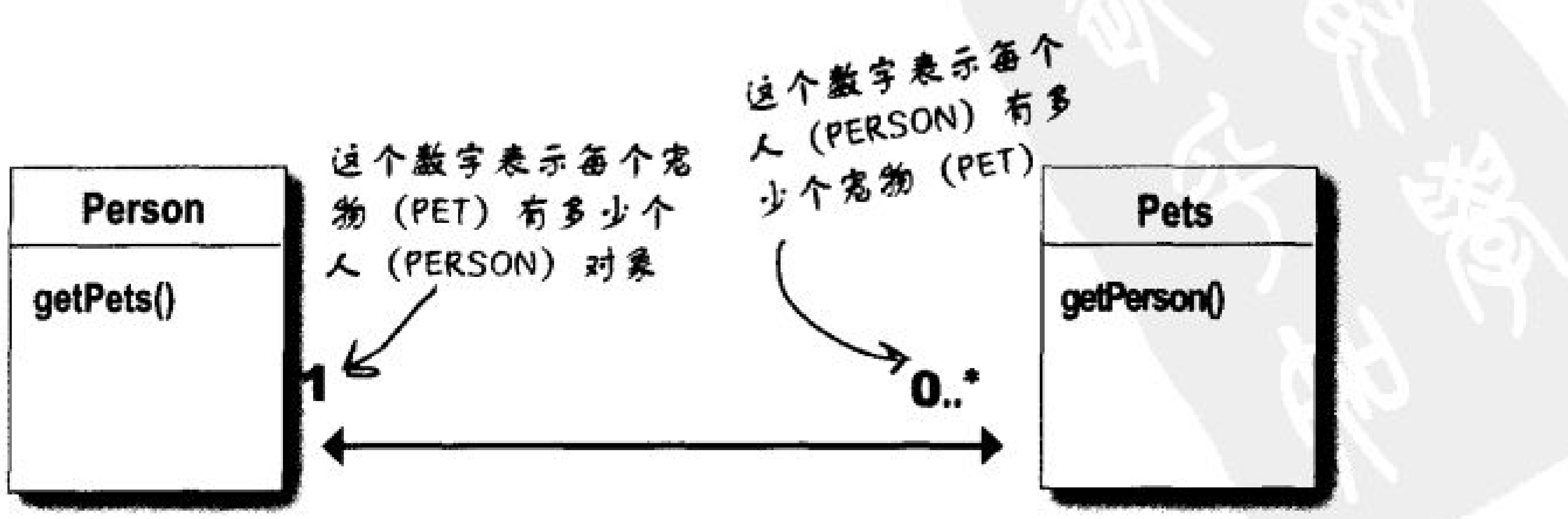
多个 Actor ←有— 一个 Movie

# bean类中的多重性

下面显示了bean代码中是怎么体现这种多重性的。你要掌握这种多重性注解，这也是考试对UML的惟一要求。



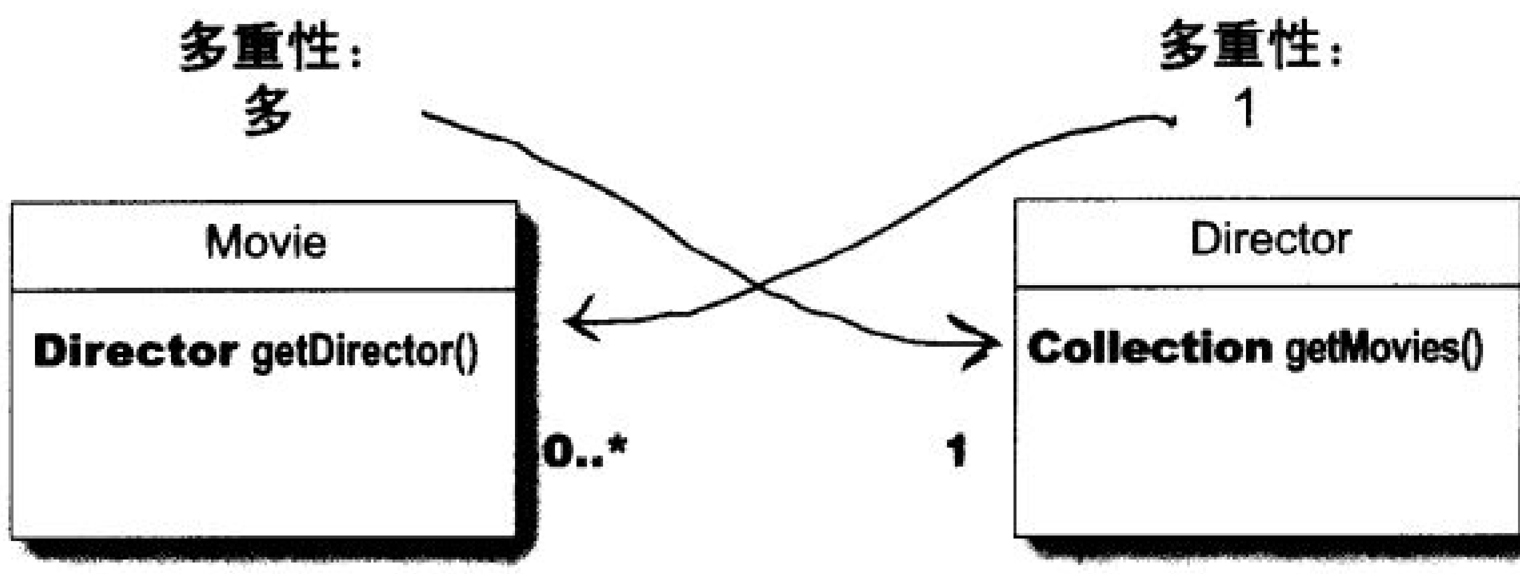

要读懂上面这个图，必须沿着箭头看它的目标。在Person与Pets的关系中，Person的多重性为ONE（1），而Pets的多重性为MANY（多，也可以是0）。要得出一个Person有多少个Pets，必须沿着箭头从Person到Pets。与类最近的数字就是这个类相对于另一个类的多重性。换句话说，就是另一个类型有多少个这种类型。



# 多重性影响返回类型！

一个Movie有一个Director。调用getDirector()时，会得到一个Director。所以getDirector()的返回类型是一个Director。

但是一个Director有多个电影，所以在调用getMovies()时，会得到Movie的一个Collection。




Movie与Director的关系中，多重性为“多”。这并不是说Movie有“多个”Director……而是说Director有“多个”Movie！

所以Director有一个getMovies()方法，它会返回Movie的一个Collection。

Director的多重性为“1”，这说明Movie只会返回一个Director。

在Movie-to-Director关系中，Movie的多重性是多，而Director的多重性是1。

多重性为1，这意味着那个与你相关的对象只有一个你。Director的多重性为1，所以与Director相关的Movie对象只返回一个Director bean（不错，从技术上讲，返回的是Director的组件接口，不过你现在应该很清楚了，我们指的就是Director的组件接口）。

多重性为多，这意味着与你相关的对象有你的一个集合（Collection）。

Movie的多重性为多，所以与Movie相关的Director对象会返回Movie bean的一个Collection。

# 为持久数据字段和关系字段定义虚字段

在上一章中，我们介绍过定义容器托管字段，要相应地放入一对抽象获取方法和设置方法。我们说过，一个容器托管字段之所以存在，只是因为你为它编写了一个获取方法和一个设置方法。容器托管关系（CMR）字段也是如此。要为之定义一对抽象获取方法和设置方法，但是这里不是设置和返回一个映射至表中某列的值，而是要设置和返回另一个实体bean的引用，或者是包含实体bean引用的一个Collection。CMR字段的限制在于，它们只能表示实体bean的本地组件接口，如果方法返回一个Collection，那么声明类型只能是Collection或Set（不能是Map、List等等）。

这里的术语有点让人混淆，因为不管是CMP还是CMR字段都是容器托管持久存储字段。可以把它们分别叫作CMPCD（container-managed persistent column data，容器托管持久存储列数据）字段和CMPR（container-managed persistent relationship，容器托管持久存储关系）字段。当然，持久存储并不仅限于关系数据库，所以这里“列”这个词用得不太严格。不过，在EJB 1.1中，所有容器托管持久存储字段都是CMP字段，因为在EJB 1.1 CMP中还没有持久关系的概念。

**CMP字段和CMR字段之间惟一的差别在于类型。**

**CMR字段总是另一个实体bean的本地接口类型，或者是bean本地接口的一个Collection。**

**如果是一个Collection，则必须是java.util.Collection或者java.util.Set。**

**对于每个CMP字段（列值）和每个CMR字段（与另一个实体的关系）都要有一对获取方法和设置方法。**

### ① CMP字段

```
public abstract String getTitle();
public abstract void setTitle(String g);
```

确保参数和返回类型匹配

### ② CMR（关系）字段

```
public abstract Director getDirector();
public abstract void setDirector(Director d);
```

Director必须是Director bean的本地组件接口。

CMR“虚字段”的参数和返回类型总是另一个实体bean的本地（接口）类型，或者是一个Collection。

# 定义“抽象持久存储模式”（虚字段还不够）

要定义持久字段和关系，需要创建一个抽象持久存储模式。定义虚字段还不够。抽象持久存储模式是bean类中虚字段的组合，再加上在部署描述文件中写的一些东西。

在部署描述文件中描述CMP字段相当简单，也很直接。对于CMR，则要多做一点工作，这包括要描述关系双方各自的多重性。

不过，我们过一会再看CMR字段。现在先从CMP字段开始介绍。

**抽象持久存储模式是部署描述文件中一些东西的组合，再加上你的抽象获取方法和设置方法。**

**加在一起，就能告诉容器如何管理bean的持久存储，这包括持久字段（称为CMP字段）和持久关系（称为CMR字段）。**

### 容器需要这两样东西才能知道你有一个CMP字段“title”

#### ① 部署描述文件中

```
<entity>
   <ejb-name>MovieBean</ejb-name>
   <local-home>headfirst.MovieHome</local-home>
   <local>headfirst.Movie</local>
   <ejb-class>headfirst.MovieBean</ejb-class>
   <abstract-schema-name>MovieSchema</abstract-schema-name>
   <cmp-field>
       <field-name>title</field-name>
   </cmp-field>
     ...
<entity>
```

#### ② bean类中

```
public abstract String getTitle();
public abstract void setTitle(String g);
```

# 部署描述文件中的持久CMP字段

```
<entity>
    <display-name>MovieBean</display-name>
    <ejb-name>MovieBean</ejb-name>
    <local-home>headfirst.MovieHome</local-home>
    <local>headfirst.Movie</local>
    <ejb-class>headfirst.MovieBean</ejb-class>
    <persistence-type>Container</persistence-type>
    <prim-key-class>java.lang.String</prim-key-class>
    <reentrant>False</reentrant>
    <cmp-version>2.x</cmp-version>
    <abstract-schema-name>MovieSchema</abstract-schema-name>
    <cmp-field>
      <field-name>genre</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>movieID</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>year</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>title</field-name>
    </cmp-field>
    <primkey-field>movieID</primkey-field>
  ...
<entity>
```

> 告诉容器，正在由它（容器，而不是bean）来管理你的持久存储

> 表示主键的字段的完全限定类名

> 对你的模式命名，这样你就知道它表示哪一个实体类型。这个名字在部署描述文件中必须惟一。

> 既然已经告诉容器要管理哪些字段，还要告诉它这些字段中哪一个最特别……也就是主键字段。

**必须遵循命名约定。**

必须有一对抽象获取方法和设置方法，在get和set之后的第一个字母要大写，但是在部署描述文件中的CMP字段中，要以一个小写字母开头，而且没有get和set。也就是说，在部署描述文件中定义CMP字段时，就好像它是一个真正的实例变量一样，要遵循原来非EJB的JavaBeans命名约定。

## there are no Dumb Questions

**问：** **容器怎么知道持久字段的类型？我看到了主键类型的定义（<prim-key-class>），但是其他CMP字段的定义我没有看到。**

**答：** 容器是根据bean类中抽象获取方法和设置方法的返回类型和参数来得出的。它们最好是匹配的！如果有一个getLimitNum()返回一个int，那最好有一个取int为参数的setLimitNum(int i)。

# 在代码中使用关系

在bean代码中，使用虚关系字段的做法与使用其他获取方法和设置方法没有两样。与使用一个真正的字段（也就是类中声明的一个实例变量）相比，使用虚字段惟一的区别在于，只能通过调用抽象获取方法和设置方法（这是容器在部署时实现的）来访问虚字段。

对于关系字段，只增加了一个限制，这就是关系字段只能是一个本地组件接口类型！ 不可能有一个使用bean远程接口的CMR关系字段！

使用虚关系字段与使用其他虚字段是一样的……要通过调用抽象获取方法，尽管这些方法是你定义的，却是由容器实现的。

关系字段的类型必须是实体的本地组件接口！

CMR字段总是一个本地组件接口类型。

### bean类中的虚字段

```
public abstract Director getDirector();
public abstract void setDirector(Director d);
```

CMR "虚字段"的参数和返回类型总是另一个实体bean类型，而且必须是一个本地接口！！虚字段不能是实体的远程接口。

### bean类中展现的业务方法

```
public Director getMovieDirector() {
    return getDirector();
}
```

现在可以调用抽象获取方法向客户返回Director。

```
public String getMovieDirectorName() {
    return getDirector().getDirectorName();
}
```

如果不想把实际的Director引用暴露给客户，还可以代表客户做这个工作。实际上，我们就打算这么做，因为Director是一个本地接口，但是Movie要有远程客户。

# 抽象持久存储模式中(部署描述文件中)定义关系

CMP字段定义相当简单。只需声明一个<cmp-field>，再给出一个字段名，这个名字要与你的某个获取方法/设置方法对匹配（遵循命名规则，去掉“get”和“set”，而且CMP字段名首字母要小写）。

但CMR要复杂一些。你不能只和自己有关系，所以关系往往包括两个bean！

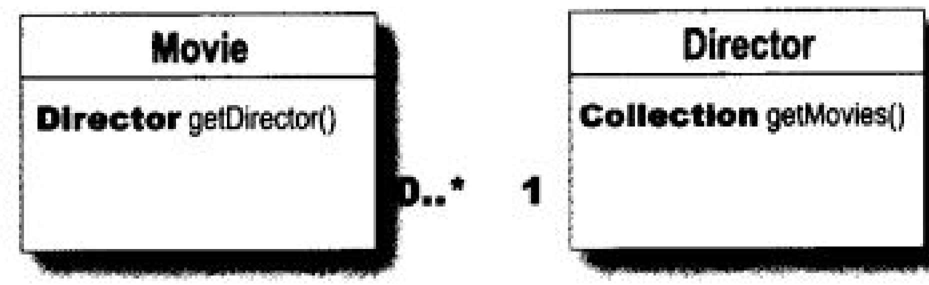
**① 在两个bean之间定义一个ejb-relation** ← 每个关系都有两个bean。

- **Ⓐ 为一个bean定义一个ejb-relationship-role**
  - **① 这个bean的多重性** ← 这个bean的多重性（也就是说，另一方有多少个这种类型的bean）
  - **② 这个bean的源** ← 我们说的是哪个bean？
  - **③ 这个bean的cmr-field** ← 必须与bean类中的“虚字段”匹配
  - **④ 这个bean的cascade-delete** ← 只有当这个bean想在对方删除时也被删除，此时才设置级联删除。

- **Ⓑ 为第二个bean定义一个ejb-relationship-role**
  - **① 这个bean的多重性**
  - **② 这个bean的源**
  - **③ 这个bean的cmr-field**
  - **④ 这个bean的cascade-delete**

关系中的第二个bean也做同样的一些设置

# Director-to-Movie关系的关系定义




## ❶ 在两个bean之间定义一个ejb-relation

```
<ejb-relation>
```

每个<ejb-relation>描述了一对bean……一个关系。实际上应该取名为<ejb-relationship>（而不是<ejb-relation>）。

### ❷ 为一个bean定义一个ejb-relationship-role

```
<ejb-relationship-role>
    <ejb-relationship-role-name>DirectorBean</ejb-relationship-role-name>
```

这是一个人造的名字！这要看你想把这个角色叫做什么。可以称之为“DirectorRole”，或者任何别的名字，只要能告诉我们所谈论的对方是谁就行。

#### ❶ 这个bean的多重性

```
<multiplicity>One</multiplicity>
```

要记住，这不是说Director bean有多少个对方……而是说另一方有多少个Director bean。Movie只有一个Director。

#### ❷ 这个bean的源

```
<relationship-role-source>
    <ejb-name>DirectorBean</ejb-name>
</relationship-role-source>
```

可以使用喜欢的任何角色名，不过容器早晚都要知道我们说的到底是哪个bean。把bean的<ejb-name>（这只是一个标签，告诉容器到哪里去找这个bean的实际定义）放到部署描述文件的<enterprise-beans>部分中。它必须与部署描述文件中的一个<ejb-name>匹配。

#### ❸ 这个bean的cmr-field

```
<cmr-field>
    <cmr-field-name>movies</cmr-field-name>
```

这必须与bean类中对应getMovies() / setMovies()的一个“虚字段”匹配

```
    <cmr-field-type>java.util.Collection</cmr-field-type>
</cmr-field>
```

只有当另一方与这个bean有一个“多”关系，才使用<cmr-field-type>，这说明你要保存多个bean，所以需要一个Collection。还可以用java.util.Set（Collection和Set是惟一的两个选择）。

#### ❹ 这个bean的cascade-delete

（这个bean未指定）

```
</ejb-relationship-role>
```

## ❺ 为第二个bean定义一个ejb-relationship-role

```
<ejb-relationship-role>
    <ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
```

记住，这只是你自己（或者应用组装人员或某个工具）使用的名字。它并不表示Java代码里的什么东西。

### ❶ 这个bean的多重性

```
<multiplicity>Many</multiplicity>
```

这说明Director会有多个这种类型（Movie）。所以Director会有一个类型为Collection（而不是单个的Movie）的CMR字段。

### ❷ 这个bean的源

```
<relationship-role-source>
    <ejb-name>MovieBean</ejb-name>
</relationship-role-source>
```

这必须与部署描述文件<enterprise-beans>部分中的<ejb-name>匹配（它指出了Movie实体bean是在哪里定义的）。

### ❸ 这个bean的cmr-field

```
<cmr-field>
    <cmr-field-name>director</cmr-field-name>
</cmr-field>
```

请看……这里没有类型！这个字段没有一个<cmr-field-type>，这是因为它不是一个Collection。实际上，<cmr-field-type>标记存在的惟一原因只是用于字段类型是Collection的时候，而且必须区别Set还是Collection。

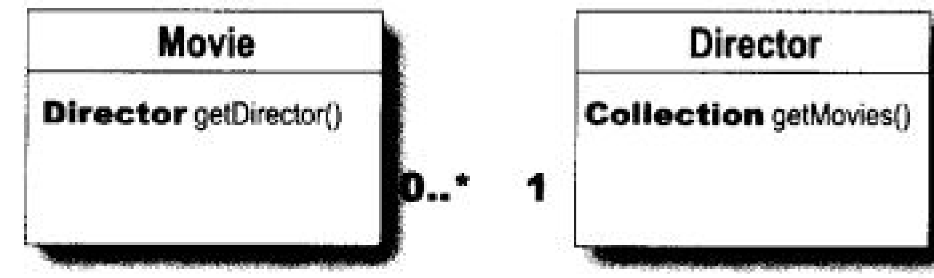
### ❹ 这个bean的cascade-delete

```
    <cascade-delete />
  </ejb-relationship-role>
</ejb-relation>
```

如果希望在对方删除时也把这个bean删除，就放上这个标记。

## Sharpen your pencil

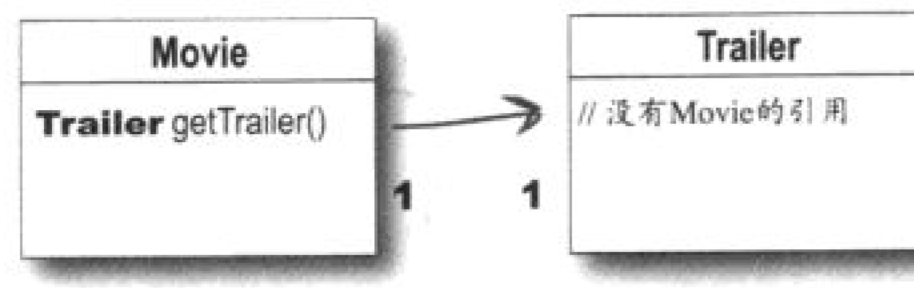
这是Director-to-Movie关系的部署描述文件，不过这里少了一些东西。看看你能不能不看前面几页就正确地把这些空填上！




## Director-to-Movie关系

```
<relationships>
   <ejb-relation>
     <ejb-relationship-role>
       <ejb-relationship-role-name>_______________ </ejb-relationship-role-name>
       <multiplicity>________</multiplicity>
       <relationship-role-source>
         <ejb-name>DirectorBean</ejb-name>
       </relationship-role-source>
       <cmr-field>
         <cmr-field-name>__________ </cmr-field-name>
         ___________________________________________
       </cmr-field>
     </ejb-relationship-role>
     <ejb-relationship-role>
       <ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
       <multiplicity>_________</multiplicity>
       <cascade-delete />
       <relationship-role-source>
         <ejb-name> ___________</ejb-name>
       </relationship-role-source>
       ___________________________________________
       ___________________________________________
       ___________________________________________
     </ejb-relationship-role>
   </ejb-relation>
</relationships>
```




## 关系可以是单向的

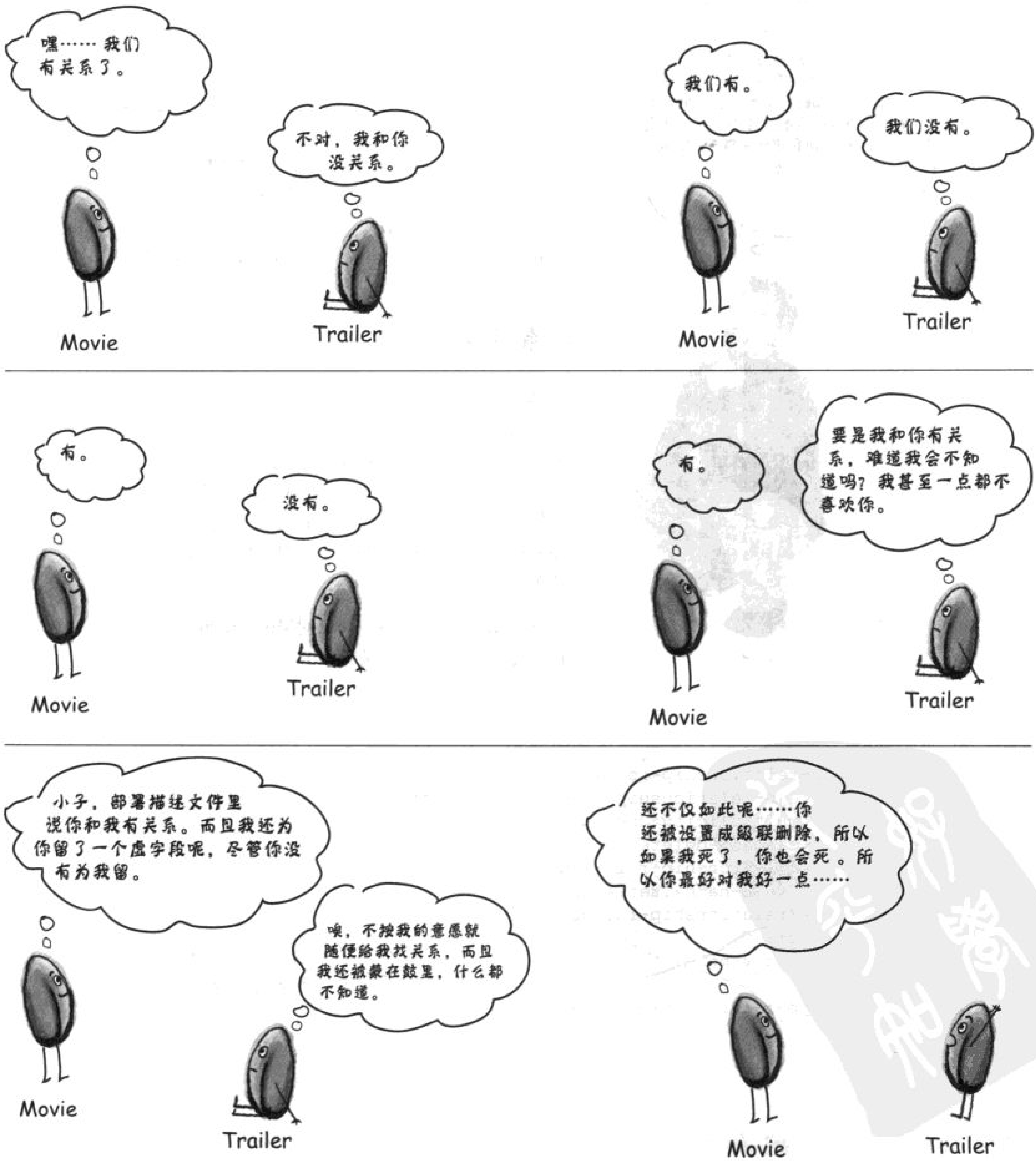
在两个bean之间有一个关系，但只是其中一个bean有一个CMR字段，这是可以的。例如，如果在一个Movie和它的Trailer之间建立了一个关系（这是一个一对一的关系），你不希望客户使用 Trailer来得到一个Movie，只需从Trailer bean中把对应Movie的CMR字段去掉。就这么简单。

在这种情况下，TrailerBean对MovieBean一无所知，尽管它们确实有关系。

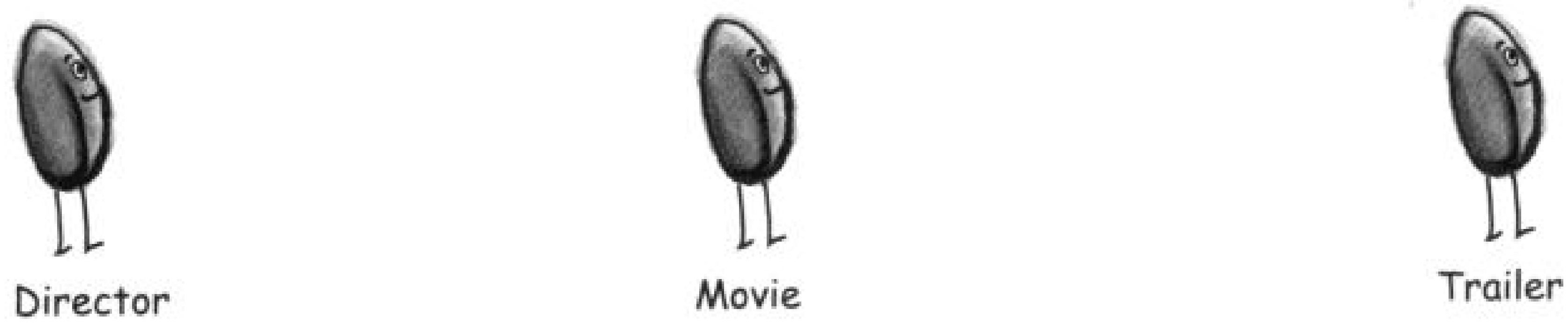
```
<ejb-relationship-role>
    <ejb-relationship-role-name>TrailerBean</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <cascade-delete />
    <relationship-role-source>
      <ejb-name>TrailerBean</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name></cmr-field-name>
    </cmr-field>
 </ejb-relationship-role>
```

如果不希望这个bean有对方的一个引用，可以在关系角色中去掉cmr-field。

## 单向关系不好的一面


嘿……我们有关系了。
不对，我和你没关系。
我们有。
我们没有。
Movie
Trailer
Movie
Trailer
有。
没有。
有。
要是我和你有关系，难道我会不知道吗？我甚至一点都不喜欢你。
Movie
Trailer
Movie
Trailer
小子，部署描述文件里说你和我有关系。而且我还为你留了一个虚字段呢，尽管你没有为我留。
噢，不按我的意愿就随便给我找关系，而且我还被蒙在鼓里，什么都不知道。
还不仅如此呢……你还被设置成级联删除，所以如果我死了，你也会死。所以你最好对我好一点……
Movie
Trailer
Movie
Trailer

# 级联删除可以传播




## Director-to-Movie关系

**Director（不能级联删除）**

**Movie - <cascade-delete/>**

在Director-to-Movie关系中，不能对Director bean设置级联删除，因为Movie的多重性为“多”。想想吧……如果有人删除了一个Movie，Director也要被删除吗？当然不是！这个Director 还可能是其他电影的导演，而其他电影可能仍然存在。

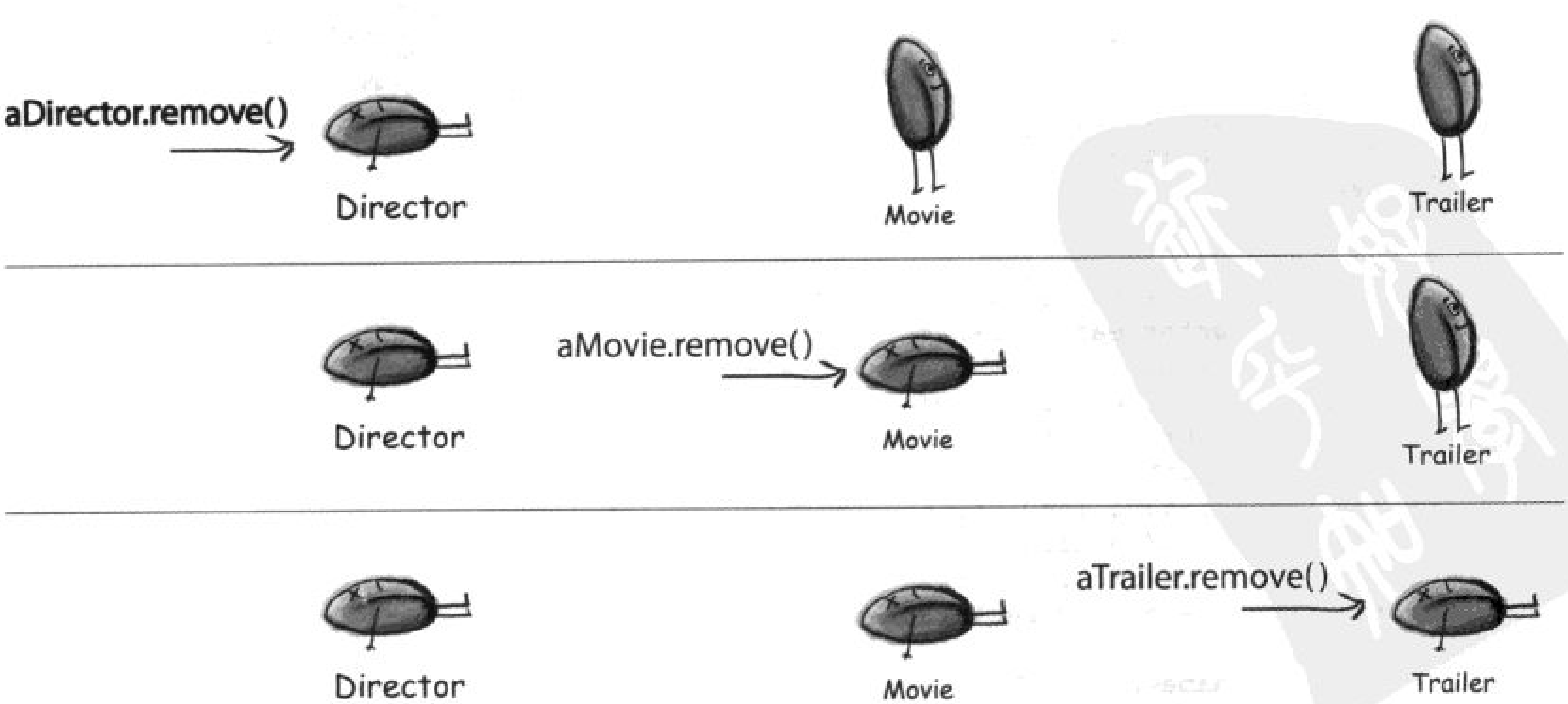
但是……Director对Movie的多重性为“1”，所以Movie bean可以说：“我只能有一个Director，所以如果他没了，我也没了。”

## Movie-to-Trailer关系

**Movie -（可以级联删除，但是不想这样做）**

**Trailer- <cascade-delete/>**

因为这是一个一对一关系，所以关系双方都可以设置一个级联删除。但是我们只希望从一个方向上级联删除，也就是删除了Movie就会删除Trailer，但反之不然。删除了Trailer并不意味着我们希望Movie也被删掉，不过这由我们的业务逻辑说了算，而不是EJB规则决定的。



# 抽象模式在<entity>元素中，而<ejb-relation>在<relationships>元素中

```
<ejb-jar>
  <display-name>MovieJar</display-name>
  <enterprise-beans>
    <entity>
      <display-name>MovieBean</display-name>
      <ejb-name>MovieBean</ejb-name>
      <local-home>headfirst.MovieHome</local-home>
      <local>headfirst.Movie</local>
      <ejb-class>headfirst.MovieBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>MovieSchema</abstract-schema-name>
      <cmp-field>
        <field-name>genre</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>movieID</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>year</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>title</field-name>
      </cmp-field>
      <primkey-field>movieID</primkey-field>
      <security-identity>
           <use-caller-identity></use-caller-identity>
      </security-identity>
    </entity>

    <entity>
      <display-name>DirectorBean</display-name>
      <ejb-name>DirectorBean</ejb-name>
      <local-home>headfirst.DirectorHome</local-home>
      <local>headfirst.Director</local>
      <ejb-class>headfirst.DirectorBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>DirectorSchema</abstract-schema-name>
      <cmp-field>
        <field-name>name</field-name>
      </cmp-field>
```

持久字段的抽象模式在<entity>元素中定义，这个元素位于<enterprise-beans>元素的下面，所有bean都在<enterprise-beans>元素中描述。

哇！注意看少了什么！！这里没有对应Director的<cmp-field>。<entity>部分里没有列出<cmr-field>元素。它们的定义只放在<relationships>部分（下一页）中。

```
    <cmp-field>
      <field-name>winner</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>directorID</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>degrees</field-name>
    </cmp-field>
    <primkey-field>directorID</primkey-field>
    <security-identity>
      <use-caller-identity></use-caller-identity>
    </security-identity>
  </entity>
</enterprise-beans>

<relationships>
  <ejb-relation>
```

<enterprise-beans>部分（所有bean都已经描述）之后，接下来是一个<relationships>部分，它使用<ejb-name>元素指回<enterprise-beans>部分。

```
    <ejb-relationship-role>
      <ejb-relationship-role-name>DirectorBean</ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source>
        <ejb-name>DirectorBean</ejb-name>
      </relationship-role-source>
```

这个名字必须与此部署描述文件中的某个东西匹配，也就是<enterprise-beans>部分中<entity>元素所描述的实体

```
      <cmr-field>
        <cmr-field-name>movies</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
```

只有当对方（不是你）的多重性是"多"时才有<cmr-field-type>，这只能是Collection或Set（不能是List、Map之类）。

```
    <ejb-relationship-role>
      <ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <cascade-delete />
      <relationship-role-source>
        <ejb-name>MovieBean</ejb-name>
      </relationship-role-source>
```

MovieBean说："你删除我的Director时可以把我也删掉。"

```
      <cmr-field>
        <cmr-field-name>director</cmr-field-name>
      </cmr-field>
    </ejb-relationship-role>

  </ejb-relation>
</relationships>
```

没有<cmr-field-type>，因为在这个关系中Director的多重性是"1"，所以只返回一个Director，而不是Director的Collection。

# 带CMR字段的MovieBean代码

```
package headfirst;

import javax.ejb.*;
import javax.naming.*;
import java.util.*;

public abstract class MovieBean implements EntityBean {

   private EntityContext context;

   public String ejbCreate(String movieID, String title, int year,String genre,
                              String directorID) throws CreateException {

     setMovieID(movieID);
     setTitle(title);
     setYear(year);
     setGenre(genre);
     return null;
   }
```

在这里设置持久字段（通过调用抽象设置方法）。现在还不能尝试引用我们的持久关系，必须等到已经插入之后才可以，而这要在ejbCreate()完成之后才会发生。

```
   public void ejbPostCreate(String movieID, String title, int year, String genre,
                              String directorID) throws CreateException {
     try {
      InitialContext ctx = new InitialContext();
      DirectorHome dirHome = (DirectorHome) ctx.lookup( "java:comp/env/ejb/DirectorHome" );
      Director dir = dirHome.findByPrimaryKey(directorID);
      setDirector(dir);
    } catch(Exception ex)
      { //处理异常}
   }
```

现在可以安全地设置对应Director的持久关系字段了。我们先查找Director（使用从创建方法得来的directorID参数），并调用抽象设置方法来指定这个Director。

```
   public String ejbHomeListMoviesByDirectorAndGenre(String dirID, String genre) {
       String list = null;
       try {
          Collection c = ejbSelectGetMoviesByDirectorAndGenre(dirID, genre);
          Iterator it = c.iterator();
          while (it.hasNext()) {
             list += it.next();
           }
     } catch (Exception ex) { ex.printStackTrace();}
      return list;
  }
```

这是一个home业务方法，客户可以用这个方法得到满足查询的所有电影的一个列表。我们使用了一个ejbSelect方法，它返回一个String对象的Collection。通过使用选择方法，就可以让容器来编写所有数据库代码！

```
public String ejbHomeListAllMovies() {
      String list = null;
        try {
           Collection c = ejbSelectGetAllMovies();
           Iterator ita = c.iterator();
              while (ita.hasNext()) {
                 Movie movie = (Movie) ita.next();
                list += " " + movie.getMovieTitle();
              }

        } catch (Exception ex) { // 处理异常}
      return list;
  }

   public String getMovieTitle() {
      return getTitle();
   }

   public String getMovieDirectorName() {
       return getDirector().getDirectorName();
   }

   public abstract String getMovieID();
   public abstract void setMovieID(String i);

   public abstract String getTitle();
   public abstract void setTitle(String t);

   public abstract int getYear();
   public abstract void setYear(int y);

   public abstract String getGenre();
   public abstract void setGenre(String g);

   public abstract Director getDirector();
   public abstract void setDirector(Director d);

   public abstract Collection ejbSelectGetAllMovies() throws FinderException;

   public abstract Collection ejbSelectGetMoviesByDirectorAndGenre(String dir, String
genre) throws FinderException;


   public void unsetEntityContext() { }
   public void ejbLoad() { }
   public void ejbStore() { }
   public void ejbActivate() { }
   public void ejbPassivate() { }
   public void ejbRemove() { }

   public void setEntityContext(EntityContext ctx) {
      context = ctx;
   }
}
```

这是另一个home业务方法，它使用一个选择方法来找出所有电影。这个选择方法返回具体Movie对象的一个Collection，所以可以遍历这个集合，取出电影的片名，以便向客户返回一个很大的String（所有片名连在一起）。

这是组件接口展现的两个业务方法……

我们有5个虚字段：4个CMP字段和1个对应Director的CMR字段。

这两个选择方法声明为抽象方法。Collection并没有指出这些方法要返回什么……不过我们会告诉容器其中一个方法返回Movie的Collection，另一个选择方法返回String的Collection。

其他的都是常规的普通回调方法。

# MovieBean的home接口

```
import javax.ejb.*;
import java.util.*;
import java.rmi.*;


public interface MovieHomeRemote extends EJBHome {

    public MovieRemote create(String movieID, String title, int year, String genre,
                              String directorID) throws CreateException, RemoteException;

    public MovieRemote findByPrimaryKey(String key) throws FinderException,
                                                           RemoteException;

    public Collection findByGenre(String genre) throws FinderException, RemoteException;

    public String listAllMovies() throws FinderException, RemoteException;

    public String listMoviesByDirectorAndGenre(String directorID, String genre)
                                              throws FinderException, RemoteException;

    public Collection findMoviesByKevinDegrees(int degrees) throws FinderException,
                                                                   RemoteException;
}
```

哇！3个查找方法不在bean类中……要记住，容器会实现你的查找方法，你不用在bean类代码中放有关查找方法的任何东西。只能把它们放在home里。

（嗯，这也不完全正确……你还要在某个地方告诉容器怎么实现查找方法，我们会在下一页说这个问题……）

home业务方法在home接口中没有前缀，但是对应到bean类中时，都以前缀“ejbHome”开头。


容器是怎么知道如何实现选择方法的呢？
还有查找方法呢？我能理解它是怎么做findByPrimaryKey的，因为它已经知道主键，但是其他查找方法呢？
我想知道……它到底怎么知道你的哪些字段映射到这个数据库？你怎么把你的抽象模式连接到实际的数据库表和列呢？

# 从抽象模式映射到实际数据库

你必须告诉容器如何管理你的持久数据，以及在哪里管理，这是迟早的事。持久数据的管理是以一种特定于开发商的方式在部署描述文件之外发生的。当然，这就意味着考试不会考这个。每个开发商都有自己的一套机制，但是我们通常可以得到某种GUI，如果你幸运，可以通过这个GUI告诉容器连接到一个特定数据库，然后通过拖放或者画线来建立CMP字段到一个或多个表中一些列的连接。

因为这个映射发生在EJB之外，这是在服务器中完成的，所以我们没有多少可说的。我们最感兴趣的是怎么从抽象查询映射到实际的查询，而这是以一种很酷的方式完成的，这里使用了一种独立于开发商的查询语言，即EJB-QL，这是专门为实体bean查询设计的。下一页将介绍有关的内容。

```
<abstract-schema-name>MovieSchema</abstract-schema-name>

<cmp-field>
  <field-name>genre</field-name>
</cmp-field>

<cmp-field>
  <field-name>movieID</field-name>
</cmp-field>

<cmp-field>
  <field-name>year</field-name>
</cmp-field>

<cmp-field>
  <field-name>title</field-name>
</cmp-field>
<primkey-field>movieID</primkey-field>
```

请注意，我们没有映射Director ID列（这个列保存了对应Director表的外键），因为并没有为Director ID建立一个持久CMP字段。既然可以得到实际的Director（实体bean），还有什么必要这样做呢？这就引入了CMR字段，不过我们没有把这个CMR字段映射到数据库。通过在部署描述文件的<relationships>部分中把两个bean匹配起来就可以建立关系。

| MovieID | Title | Genre | DirectorID | Year |
|---|---|---|---|---|
| 12 | Crouching Pixels, Hidden Mouse | Action | 42 | 2000 |
| 5 | The Fifth Array Element | Action | 27 | 2003 |
| 22 | The Return of the Bean Queen | Fantasy | 27 | 2001 |
| 11 | Lord of the Loops | Sci-fi | 42 | 2002 |

# 为选择和查找方法编写可移植查询




EJB-QL是一种可移植的查询语言，利用它，你就可以在部署描述文件中编写类似于SQL的语句！只需把它们放在XML中，而不必对实际的数据库有任何了解。

作为一个bean提供者，你可以自由地编写你需要的任何查询，你完全可以相信部署人员最终肯定能把你的CMP字段映射到实际的数据库表和数据。这里我们关心的只是怎么为home接口和组件接口中展现给客户的方法编写查询。

选择方法只是为了让你更轻松一些，而让容器根据你的查询来建立实际的数据库访问代码。

所以这个过程基本上是这样的：

1. **编写一个home业务方法。**
2. **实现home业务方法来调用完成实际数据访问的ejbSelect方法。**
3. **在bean类中声明抽象选择方法。**
4. **为选择查询编写EJB-QL。**

# MovieBean的EJB-QL

下面来看MovieBean需要的所有查询的相应EJB-QL。如果你还不能完全理解，先别着急，在学习这些示例的过程，我们会详细地分析各个部分，并说明有关的语法。如果你懂SQL，以下大部分内容对你来说并不陌生（不过可以看到，还是有一些差别的）。如果你不了解SQL，也没关系。下面会直接写查询，而且你可以通过规范来更多地了解EJB-QL语言。考试只要求你知道基本的EJB-QL就可以了。

这个方法返回Movie对象的一个Collection

## bean类中

```
public abstract Collection ejbSelectGetAllMovies() throws FinderException;
```

## 部署描述文件中

```
SELECT OBJECT (m) FROM MovieSchema m
```

这是说"返回'm'所表示的任何东西"。在这个例子中，"m"表示MovieSchema，而MovieSchema实际上表示Movie实体bean类型。所以这个第一部分说的是"给我Movie bean"。

这就像声明一个变量……"让'm'表示Movie bean的抽象模式"。不过，实际上可以这样想，"让'm'表示Movie组件接口类型"，换句话说，也就是Movie bean。所以它的实际含义是，"让'm'表示一个Movie bean"。

## 实际含义

"给我所有的Movie bean"

# 使用一个可选的WHERE子句

这个示例要对查找返回的对象有所限制，这里增加了一个WHERE子句，指出对象必须满足哪个额外条件才堪称"能从这个查询返回的东西"。通过WHERE子句，这就是在说："不要把所有老Movie一股脑儿都给我……我只想要满足这样一些条件的电影……"

## bean类中

这个方法返回Movie对象的一个Collection

```
public abstract Collection ejbSelectGetAllMovies() throws FinderException;
```

## 部署描述文件中

"genre"必须是Movie bean中的一个CMP字段，这说明可以在bean类中找到一个setGenre() / setGenre()方法对

```
SELECT OBJECT (m)  FROM MovieSchema m  WHERE m.genre = ?1
```

这表示第1个参数

这是说我们想要"m"表示的某个bean类型的对象……

让"m"表示Movie bean类型

对返回的Movie做限制，只能是CMP字段"genre"与选择方法第1个参数相匹配的Movie才能返回。这里的"1"表示第1个参数。"m.genre"使用点导航操作符建立了一个以面向对象方式导航的"路径"。简单，左操作数是一个类型，右操作数是该类型的一个成员字段。这与常规的Java语法"myMovie.genre"几乎完全相同，在Java语法中，genre是一个实例变量（在这里genre是一个虚持久字段）。

## 实际含义

"把genre中与ejbSelectGetAllMovies方法的第1个参数相匹配的所有Movie bean给我。"

# 导航到另一个相关的bean

这是一个查找方法的示例，所以在bean类中是看不到它的（记住，你只为选择方法声明抽象方法，而不包括查找方法。容器会使用你的home接口来得出有哪些查找方法，然后由你来为这些查找方法编写EJB-QL）。

使用点操作符，不仅可以导航到bean中的一个CMP字段，还可以导航到CMR字段所引用bean中的另一个字段。换句话说，如果Movie有一个Director的引用（它们在一个关系中），就可以使用Movie来得到Director，再从Director请求它的一个CMP值。

## home接口中

```
public Collection findMoviesByKevinDegrees(int degrees) throws FinderException,
                                                              RemoteException;
```

## 部署描述文件中

m.director表示"Movie bean中的CMR字段'director'"

"degrees"是Director bean中的一个持久字段。

```
SELECT OBJECT (m)  FROM MovieSchema m   WHERE m.director.degrees = ?1
```

这是说我们想要"m"所表示的某个类型的对象……

……知道吗："m"表示的就是Movie bean

对返回的Movie有所限制，如果Movie的Director有一个CMP "degrees"字段，其值与查找方法的第1个参数匹配，只有这样的Movie才会返回。

## 实际含义

"如果Movie bean有一个Director，这个Director的degrees与客户在查找方法中设置的参数匹配，请把所有这样的Movie bean给我。"

或者

"我想拿到Movie，但是只要那些达到某个等级（Kevin Bacon相对等级为某个数）的导演执导的Movie" *

* "Six degrees from Kevin Bacon"是一个很流行的游戏，你可以选一个Hollywood影片类型，看看有多少连接让这个导演连到Kevin Bacon。"

# 选择一个值而不是整个bean

这是一个home业务方法的示例，这个方法取两个参数，而且返回的不是Movie，而是String。

## home接口中

这个方法返回一个String，这是满足查询的所有Movie的片名

```
public String ejbHomeListMoviesByDirectorAndGenre(String dirID, String genre)
```

## 部署描述文件中

这里有很大不同！我们返回的只是Movie片名，而不是Movie对象

m.genre 表示Movie对象中的“genre”CMP字段

```
SELECT m.title  FROM MovieSchema m

WHERE m.director.directorID = ?1  AND m.genre = ?2
```

这限制为，所返回的Movie要有这样一个Director，它的Director ID与方法的第1个参数相等

而且，Movie genre值要等于方法的第2个参数。

## 实际含义

“按我在方法参数中指定的导演和类别，返回满足条件的所有Movie的片名（String）。”

# EJB-QL SELECT

前面已经看到了一些查询，下面来看一些细节和规则。

## 查询域

查询域是一个简单的部署描述文件。所以这意味着这是一个ejb-jar文件。你在ejb-jar里放多少个bean，你就能查多少个bean。

## SELECT

SELECT子句指出这个查询要返回什么类型的东西。这有以下两种可能：

**①对应一个实体bean的**一个抽象模式类型

```
<abstract-schema-name>MovieSchema</abstract-schema-name>

SELECT OBJECT (m) FROM MovieSchema m
```

按正式的EJB术语（以及规范中），作为SELECT变量的抽象模式类型称为一个区间(RANGE)变量，因为它覆盖了整个实体bean类型区间。

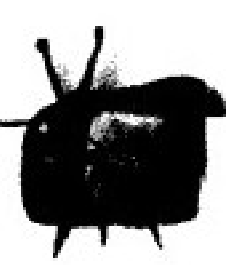
**或**

**②对应一个实体bean的<cmp-field>单值类型**

```
<cmp-field>
    <field-name>title</field-name>
</cmp-field>

SELECT m.title  FROM MovieSchema m
```

在使用<cmp-field>而非区间变量时，我们称之为单值路径表达式。



**不能使用点记法返回CMR字段!**

作为一个SELECT类型，下面这些是可以的：
m.title 或 m.genre

但不能这样：
m.director

作为SELECT类型，可以对CMP字段使用点记法，但不能用于CMR字段。

注意：这里的抽象模式名与标准规范命名约定稍有违背，因为我们想要明显地反映出在使用模式名，而不是（比如说）bean名。而根据规范，抽象模式名与组件接口名相同。

# EJB-QL SELECT — 什么时候使用Object()

OBJECT (m) 是什么?

这是J2EE设计者留的一个结构，以便同SQL的OBJECT (m)一致。

这个结构很讨厌。

不过，如果你必须返回一个bean类型而不是CMP字段类型，就必须用它。而且只要是返回CMP字段类型，**就绝对不要用它。**

如果返回bean的抽象模式类型，应使用OBJECT( )。

```
<abstract-schema-name>MovieSchema</abstract-schema-name>
SELECT OBJECT (m)
```

如果返回一个<cmp-field>，不要使用OBJECT ( )

```
<cmp-field>
   <field-name>title</field-name>
</cmp-field>
SELECT m.title
```

如果使用点号来导航路径，就不要使用OBJECT ( )

## Sharpen your pencil

圈出SELECT子句合法的EJB-QL语句

```
SELECT OBJECT (m) FROM MovieSchema m

SELECT m.title  FROM MovieSchema m

SELECT m FROM MovieSchema m

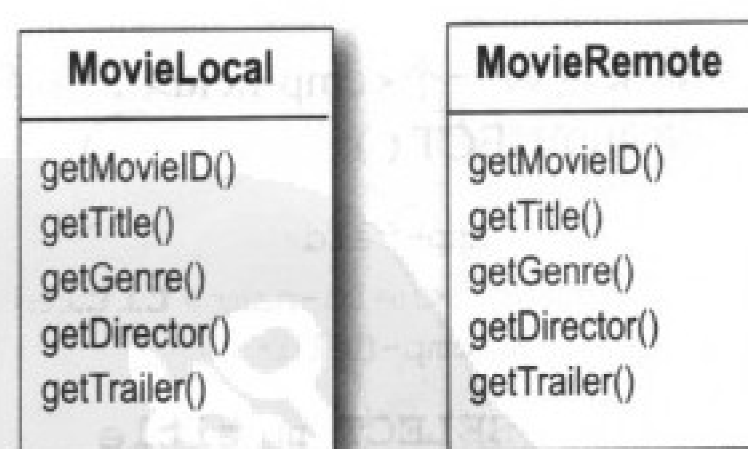
SELECT OBJECT (m.title)  FROM MovieSchema m
```

# 返回一个抽象模式类型到底是什么意思？




MovieSchema抽象模式名表示了bean类型。

就像有魔法一样，容器确实知道要返回哪个接口视图， 根据查询调用来自于一个home接口还是来自于远程接口，容器可以确定返回本地接口还是远程接口。




**看到这个：**

**MovieSchema m**

**可以这样想：**

**m = 组件接口**

（而且容器会自己分清到底是本地接口还是远程接口）

# SELECT和FROM是必须有的！

无论如何，SELECT和FROM都必须有，但WHERE子句是可选的。对于SELECT，还可以使用关键字DISTINCT指出你不希望Collection返回的结果有任何重复。

这里是使用m　　这里是声明m

```
SELECT DISTINCT OBJECT (m) FROM MovieSchema m
```

DISTINCT 是指"没有重复"

在FROM中，我们声明了标识变量"m"，并在SELECT中使用这个m。

FROM子句声明了标识变量（比如以上示例查询中的"m"）。它还定义了查询域，它告诉容器："你希望你在这里查查看……"如果你不熟悉SQL，可能觉得FROM 和 SELECT 子句出现的顺序有点奇怪。不过，可以这样想，就像有人在说："我希望你去拿点东西，我说的东西在我最上面的那个抽屉里。"

如果你需要声明更多的标识变量，以便以后在查询中使用，可以用逗号把它们分隔开：

```
SELECT OBJECT (m)
FROM MovieSchema m, DirectorSchema d
```

我们声明了两个不同的标识符，并把查询域定义为这两个抽象模式。

还可以使用AS关键字。这是可选的，不过在考试中或者其他人的代码里可能会看到这个AS，也可能你的公司在风格上有要求，必须使用AS关键字。我们都太懒了，往往都没有加AS。

```
SELECT OBJECT (m) FROM MovieSchema AS m
```

# 标识符

建立一个标识符时，必须遵循Java编程语言有关命名的原则。不能使用其他Java标识符。

### 标识符

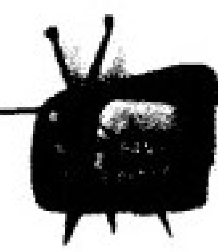
必须是合法的Java标识符。

不能与部署描述文件中的一个<abstract-schema-name>或<ejb-name>同名。

不能是EJB-QL保留字。

### EJB-QL保留字

| | |
|---|---|
| *SELECT* | *OR* |
| *FROM* | *BETWEEN* |
| *WHERE* | *LIKE* |
| *DISTINCT* | *IN* |
| *OBJECT* | *AS* |
| *NULL* | *UNKNOWN** |
| *TRUE* | *EMPTY* |
| *FALSE* | *MEMBER* |
| *NOT* | *OF* |
| *AND* | *IS* |



**当心命名冲突问题！**

要记住，在ejb-jar中，每个bean的<ejb-name>必须惟一，这说明要在一个完整的部署描述文件中惟一。<abstract-schema-name>也是如此；在部署描述文件中，每个bean的<abstract-schema-name>必须惟一。

你已经知道了，在查询中不能使用一个与部署描述文件中<ejb-name>或<abstract-schema-name>匹配的标识变量，但是要记住，EJB-QL是不区分大小写的！

所以不能这样写：

```
SELECT OBJECT (movie)
FROM Movie movie
```

在EJB-QL中，"Movie"与"movie"是一样的，而且不能有冲突的名字。

* 当前版本还没有用；将来做何使用还不确定……

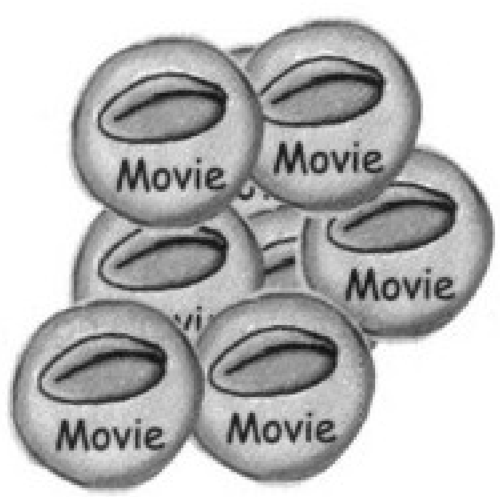
# WHERE子句

WHERE子句是可选的，它允许你对返回什么加一些额外的条件。要记住，如果没有一个WHERE子句，只要类型是SELECT使用的类型,就全部都返回。

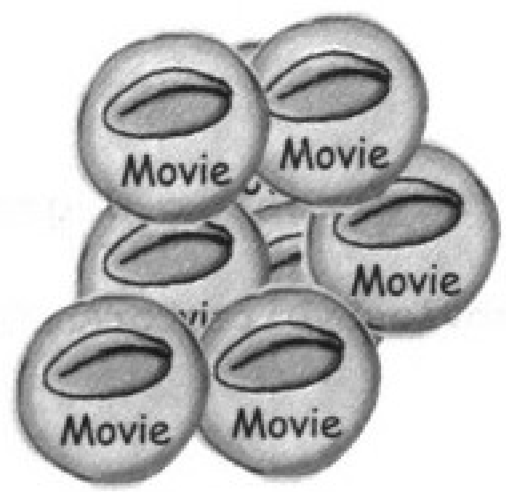
不使用WHERE

```
SELECT OBJECT (m) FROM MovieSchema m
```

现有的实体




查询返回的实体




用单引号把String直接量括起来

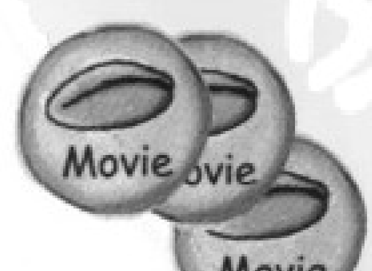
使用WHERE

```
SELECT OBJECT (m)  FROM MovieSchema m  WHERE m.genre = 'Horror'
```

现有的实体




查询返回的实体




使用WHERE，只能得到CMP字段"genre"的值为"Horror"的电影（换句话说，这些电影的setGenre()方法会返回"Horror"）。

# WHERE子句

**可以使用直接量**

```
WHERE m.director.degrees > 3
WHERE m.genre =  'Horror'
```

**可以使用输入参数**

```
WHERE m.director.degrees > ?1
```

**参数不必按顺序来……所以它们有编号!**

```
WHERE m.genre = ?2 AND m.director.degrees < ?1

public String ejbHomeListMoviesByDirectorAndGenre(String dirID, String genre)
```

**可以做比较**

```
WHERE m.director.degrees < 5
```

**但最好是同类型比较**

```
WHERE m.director < 5
```

不要这样!!

不能把Director bean与一个数字类型比较!

# 使用Collection类型的问题

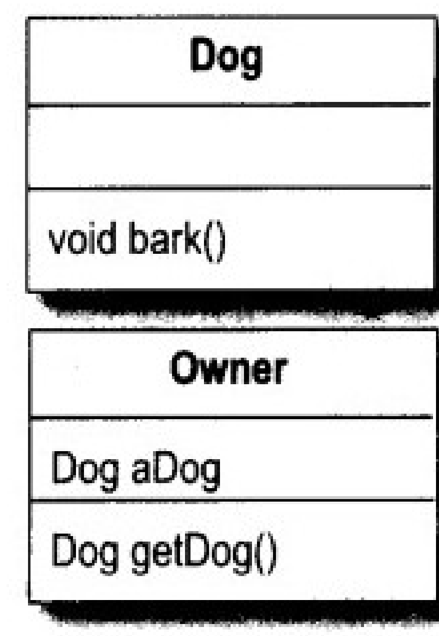
想像一下，如果你想使用DirectorSchema来返回拍恐怖片的所有导演。听上去好像很简单……Director有对应电影的一个CMR字段，每个电影又有一个对应类别的CMP字段。所以可能会写出这样的查询：

```
SELECT OBJECT (d)
FROM DirectorSchema d
WHERE d.movies.genre =  'Horror'
```

乍一看很好，但是当心！这是非法的。

为什么不能这样做？

退一步说，假设这是Java代码。你在Java中通常怎么使用点操作符呢？可能是这样：




```
Owner o = new Owner();
o.getDog().bark();
```

Owner的方法　Dog的方法

这是对的……我们可以通过Owner得到他的Dog，再在Owner的狗上调用bark()。

Owner有一个getDog()方法，而且Dog有一个bark()方法。所以这没有问题。

但是，如果修改Owner，使得一个Owner可以有多个Dog对象，会怎么样呢？这样就会……




```
Owner o = new Owner();
o.getDogs().bark();
```

Owner的方法

唉呀！！现在到底对谁调用这个方法？一个Dog？还是一个Collection……

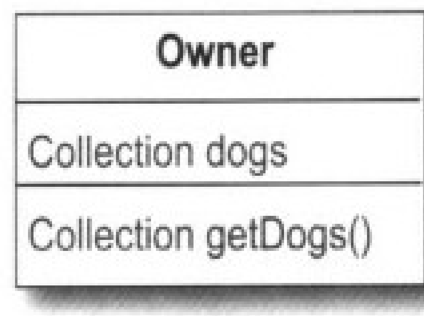
# Collection不会bark()!




如果路径中使用的某个东西是Collection，那么就不能使用点记法导航。例如，下面这样就不行：

```
Owner o = new Owner();
o.getDogs().bark();
```

这会返回一个Collection。而你不能在一个Collection上调用bark()!




只是集合中的Dog对象会bark()!所以，在Java中，必须首先访问集合中的单个Dog元素，而且一次让一个Dog叫bark()。上面的Dog代码就相当于前面写的WHERE子句：

```
WHERE d.movies.genre = 'Horror'
```

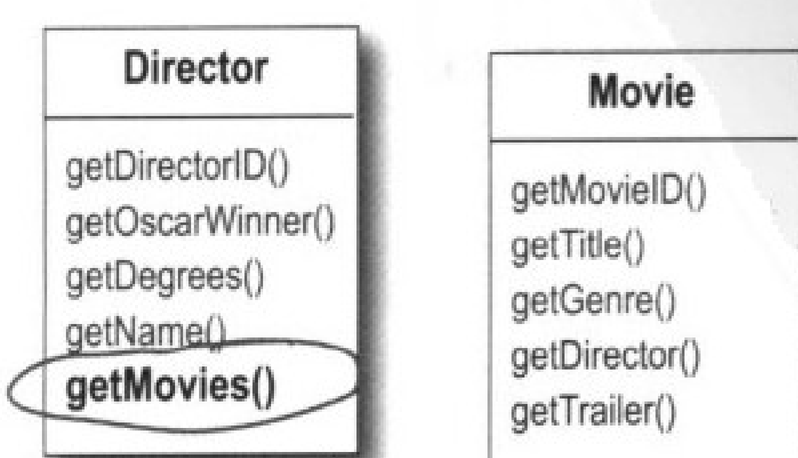
不行!!

由于“movies”和“genre”都是虚字段，有存取方法，可以把它想成是：

```
d.getMovies().getGenre() = 'Horror'
```

不行!!



# 利用IN操作符，就是在说“对于集合中的单个元素……”

在FROM子句中用IN操作符可以引用一个Collection，而命名一个标识符来表示这个Collection中的单个成员。

```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d, IN(d.movies) m
WHERE m.genre = 'Horror'
```

现在“m”表示电影Collection（这是Director的一个CMR字段）中的单个Movie

让“m”表示Collection中的单个Movie。所以，“m”不表示Collection类型，而表示Movie类型（Collection所包含的类型）。

这个查询会搜索所有Director，并搜索每个Director的每部Movie，查找一个genre与‘Horror’匹配的电影。这个查询会返回一个集合（没有重复，因为前面已经指出Director要“DISTINCT”）

**IN操作符在EJB-QL中是重载的！**

FROM子句中的IN操作符与WHERE子句中的IN比较操作符是不同的。

在WHERE子句中，可以说：
d.genre IN（‘Horror’，‘Fantasy’）

这只是以下写法的一个简写形式：
d.genre = ‘Horror’ OR d.genre = ‘Fantasy’

但是这与FROM子句中的IN操作符完全不相干，在FROM子句中IN操作符用于指定一个Collection中的单个成员，所以一定要记住：

FROM子句中，IN是指“查看Collection中的单个对象，不要把Collection用作为类型，而要使用Collection中的元素的类型。”

WHERE子句中，IN是指“看看字段值是否与这个集合中的某个直接量匹配……”

# BETWEEN表达式

假设你想找出数据库中所有20世纪60年代和70年代的电影，但又不想包括1975年或1976的电影，这两年实在没什么好电影。

可以这样说：

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE (m.year > 1959 AND m.year < 1975) OR
      (m.year > 1976 AND m.year < 1980)
```

或者，还可以用一种更好的办法：

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE (m.year BETWEEN 1960 AND 1979) AND
      (m.year NOT BETWEEN 1975 AND 1976)
```

在这个示例中，注意到BETWEEN是内包含的（也就是说，1960和1979也包括在内），而NOT BETWEEN是外包含的（即不含区间边界）。



# 其他"IN"

在FROM子句中我们用过IN来帮助在集合中导航，但是IN还有另外一重身份，可以说IN在WHERE子句中有所重载。假设你想从数据库中各类影片的一个子集中选择电影：

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.genre IN('horror', 'mystery')
```

这个查询只会返回类别是"horror"或"mystery"的电影。利用这种IN，可以创建一种类似于枚举表的东西。

通常还可以用NOT IN来做些工作：

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.genre NOT IN('romance', 'comedy')
```

这个查找只会返回类别不是"romance"或"comedy"的电影

**"!="不同于"< >"**

在WHERE子句中比较值时，"< >"是指"不等于"。此时要使用"< >"而不是"!="。但是从Java来看，好像"!="也没有什么不妥……但是千万不要上当。

**String直接量要放在单引号里！**

在讨论Java语法与EJB-QL的不一致时，不要被双引号里的String搞糊涂。在EJB-QL中这是错的：
WHERE m.foo IN( "bar", "baz" )
而应当是( 'bar', 'baz' )

# IS EMPTY比较表达式

用它你能知道当前集合是否为空：




```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d
WHERE d.movies IS EMPTY
```

这个查询只返回没有拍出任何电影的导演。

# IS NOT EMPTY比较表达式

IS NOT EMPTY就是说“不要给我这个字段为空的东西……”

```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d
WHERE d.movies IS NOT EMPTY
```

这个查询只返回至少拍过一部电影（作品集不为空）的导演。

# LIKE表达式

Like表达式用于对一个值路径表达式和一个String直接量进行比较。LIKE的威力在于，String直接量可以有通配符元素，这就给了你一个简单的模式匹配器。“%”通配符与0个或多个字符匹配；“_”只与单个字符匹配。

```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d
WHERE d.phone LIKE '719%'
```

“%”符号是一个通配符，与0个或多个字符匹配。

这个查询只返回电话号码以数字“719”开头的导演（嘿，电话号码可能是String，因为里面有一些短线……）。

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.filmcode LIKE '7_mm'
```

这个查询会查找片号为7Xmm系列的电影。

“_”符号是一个通配符，表示单个字符

# NOT LIKE表达式

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.filmcode NOT LIKE '%mm'
```

你可能能猜出来这个查询是做什么的……它会返回片号不以“mm”结尾的电影。

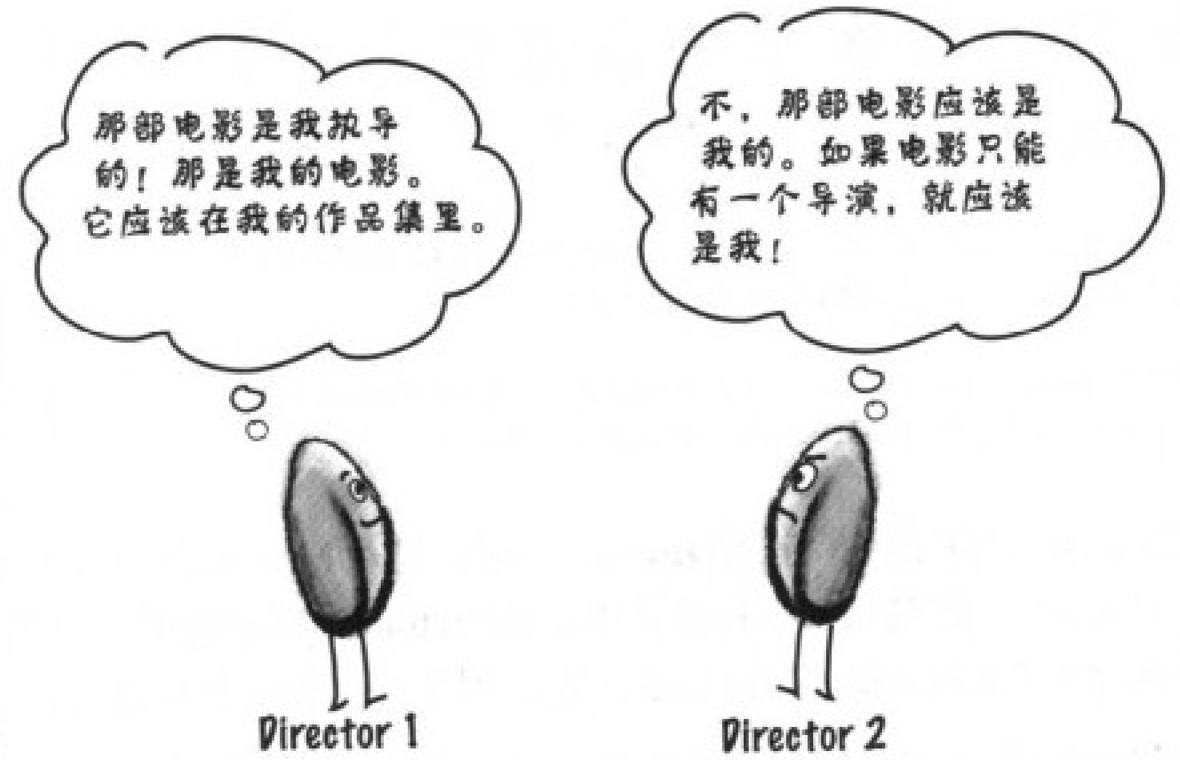
## 要点

- 实体bean可以与其他实体bean有持久关系。
- 在bean类中，容器托管关系（CMR）字段的定义与CMP字段的定义是一样的，也是用一对抽象获取方法和设置方法来定义，CMR字段会返回bean的本地组件接口或者返回一个Collection。
- 如果虚字段是一个Collection，则必须声明为java.util.Collection或java.util.Set。其他Collection类型都不用声明为CMR字段的返回类型。
- 关系有多重性，可以是一对一、一对多或多对多（多对一与一对多实际上是一样的；只是看你从哪个角度来看）。
- 例如，Movie bean与一个Director bean有一个多对一关系。每个Movie只有一个Director，但是一个Director可以有多个Movie。
- 多重性影响着虚字段的返回类型。Movie只有一个Director，所以这个bean里的CMR字段就是getDirector()，它返回Director本地组件接口的一个引用。但是一个Director有多个电影，所以Director的虚字段是getMovies()，它返回一个Collection。
- CMP bean必须在部署描述文件中定义一个"抽象持久存储模式"，其中列出bean的各个CMP字段，还要标识出其中哪个字段是主键（除非是一个复合主键）。部署描述文件一定要定义主键的类型，即使主键并非bean的一个字段也需要定义它的类型（要记住，如果主键不是bean的一个字段，则必须是由bean的CMP字段所组成的一个主键类）。
- 关系（CMR）字段不在部署描述文件的<enterprise-bean>部分中定义（CMP字段是在这里定义的），而要在部署描述文件的<relationships>部分中定义。
- 每个关系都必须有两方，每一方分别在一个<ejb-relationship-role>元素中描述，其中包括CMR字段名、源（<ejb-name>）、多重性（也就是说，另一方有多少个这个bean），还有一个可选的<cascade-delete/>（这表示"如果对方被删除了，也把我删掉"）。
- 关系可以是单向的，也可以是双向的。单向关系与双向关系在部署描述文件中都以同样的方式描述，只不过单向关系中只有一方有一个CMR字段。这表示bean A有bean B的一个引用（它们有关系），但是bean B没有bean A的引用。方向性并不影响多重性或级联删除；它只是表示其中只有一方有对方的一个引用。对于容器来说，它们仍是有关系的。
- 级联删除告诉容器，如果一个bean有<cascade-delete/>标记，当对方被删除时，也把它删掉。只有当bean的多重性为1时这才可行（如果一个Director的某个Movie删掉了，你可能并不想把这个Director也删除；但是如果Movie惟一的Director删除了，则可能希望把这个Movie也删除）。
- bean不能在ejbCreate()访问自己的CMR字段；必须等到ejbPostCreate()才可以给CMR虚字段分配一个具体的值。换句话说，在ejbCreate()完成之前，不能使用CMR字段的抽象获取方法和设置方法。
- EJB-QL是一种可移植语言，可以用于指定查找方法和选择方法的查询。
- 在一个EJB-QL查询中，必须有SELECT和FROM子句；WHERE子句是可选的。
- SELECT可以返回一个<abstract-schema-type>（这实际上表示bean的本地组件接口类型），或者返回一个单值字段。
- 如果使用点记法来进行路径导航（例如m.genre），不能将Collection类型用作为路径中的一部分。要把一个Collection类型用作FROM子句的一部分，必须用IN(d.movies)操作符把它括起来。

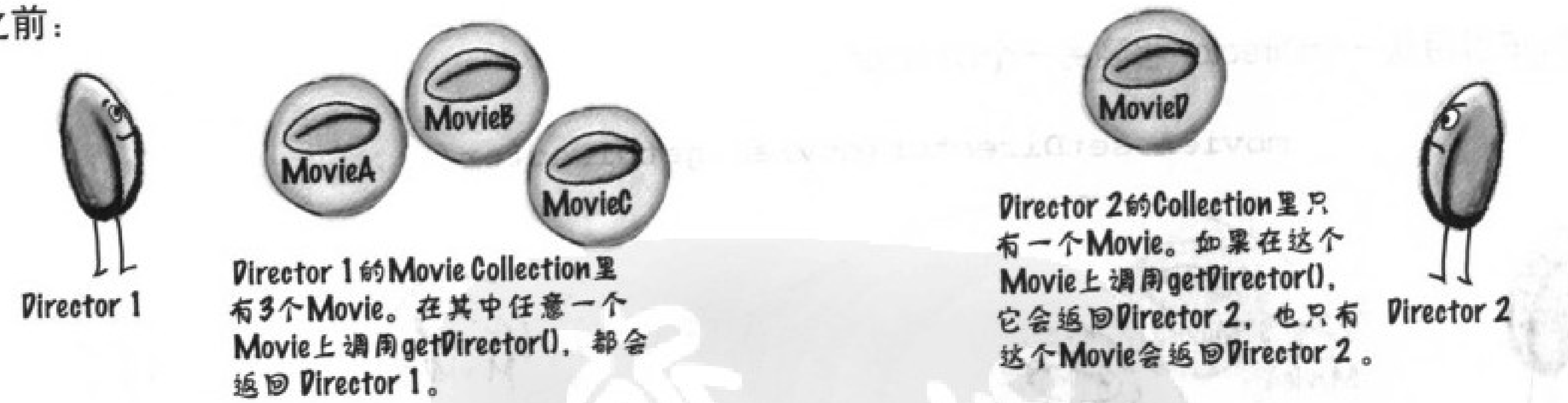
## 关系指派

一个Movie只能有一个Director。如果把一个Movie的Director字段设置为Director 1，这个Movie就会在Director 1的Movie作品集中，而不在其他Director的作品集中。如果后来又重新指定了这个Movie的Director字段，这样该Movie有了另一个Director——Director 2，那么前一个Director (Director 1) 的Collection中就不再有这个Movie了。

这一切都是自动进行的！要维护数据库的完整性，容器对关系的双方时时刻刻都在严加管理。



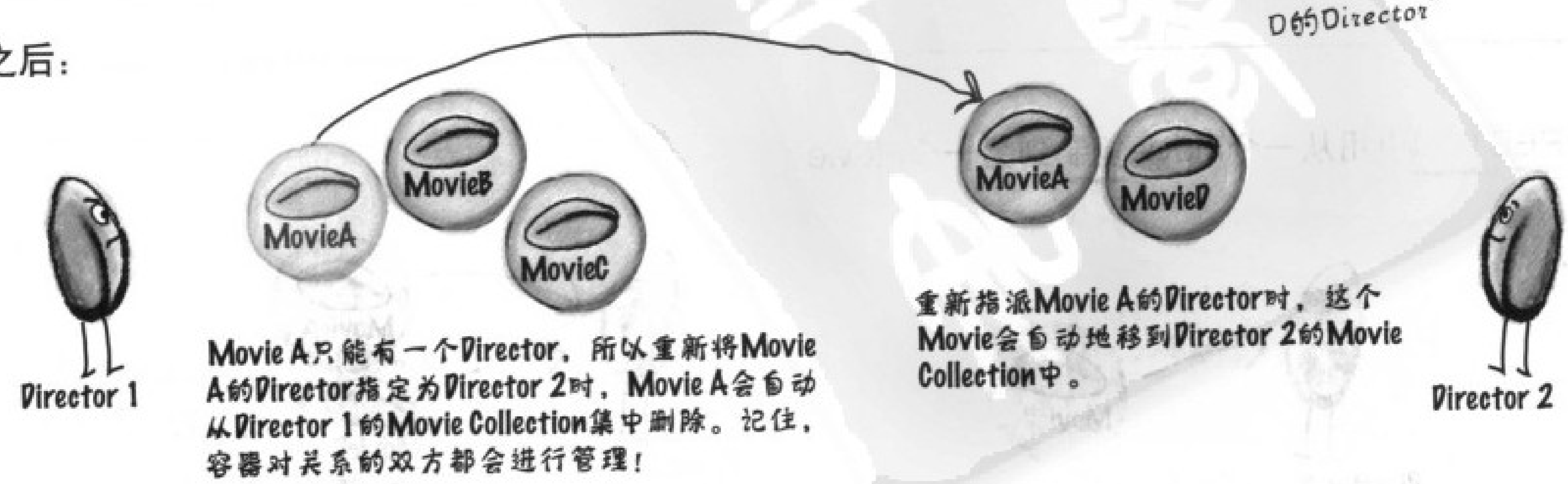

之前：




```
movieA.setDirector(movieD.getDirector());
```

这是说"把Movie A的Director设置成Movie D的Director"

之后：



# 如果关系字段的多重性是“1”，则会移走 如果多重性是“多”，则会复制

```
<ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
<multiplicity>Many</multiplicity>

<ejb-relationship-role-name>DirectorBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>
```

把Movie A的导演指定为Director 2时，这个Movie从Director 1的Collection移到Director 2的Collection。在与Movie的关系中，Director CMR字段的多重性为1，所以Movie不能同时存在于两个不同Director的Collection中，因为一个Movie只能有一个Director。

尽管我们移走了Movie，但这并不表示Director也会移动。Movie的多重性是多，所以将一个Movie的Director指派给另一个Movie时，会复制这个Director引用，这两个Movie现在就都有了一个引用。

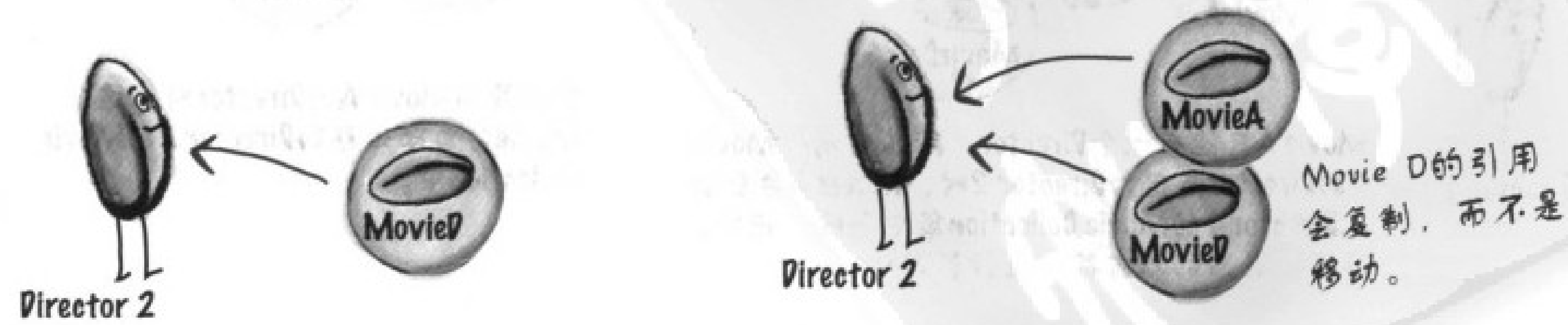
## MOVIE引用从一个Director 移到另一个Director

```
movieA.setDirector(movieD.getDirector());
```




因为Director-to-Movie关系是一对多关系，一个Movie不可能存在于多个Director的Collection中。如果修改了一个Movie的Director，这个Movie不会复制！只是会移到新Director的Collection中。

## DIRECTOR引用从一个Movie复制到另一个Movie






## 多重性和指派

给定以下双向关系：

```
<ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>

<ejb-relationship-role-name>TrailerBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>
```

这是当前场景：




画出运行以下代码之后的图：

```
movie1.setTrailer(movie2.getTrailer());
```

在这里画图……

回答以下问题：

**movie1.getTrailer()** 的结果是什么：

**movie2.getTrailer()** 的结果是什么：

**trailerA.getMovie()** 的结果是什么：

**trailerB.getMovie()** 的结果是什么：




## 多重性和指派

**给定以下双向关系：**

```
<ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>

<ejb-relationship-role-name>TrailerBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>
```

**这是当前场景：**




**画出运行以下语句之后的图：**

```
movie1.setTrailer(movie2.getTrailer());
```




**回答以下问题：**

**movie1.getTrailer()** — 返回TrailerB。

**movie2.getTrailer()** — 返回null！Movie的多重性为1，所以Trailer只能有一个Movie。一个Trailer（TrailerB）不能同时对应Movie 1和Movie 2。

**trailerA.getMovie()** — 也返回null…… Trailer的多重性为1，所以每个Movie只能有一个Trailer。TrailerB到Movie 1之后，TrailerA会被踢出来，脱离关系。

**trailerB.getMovie()** — 返回Movie1，现在它已经指派给Movie1，容器会更新Trailer的setMovie()返回的字段。

# COFFEE CRAM 模拟测验

**1** 对于实体bean提供者使用容器托管持久存储来创建持久关系，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 这个bean必须有本地接口来与另一个实体bean建立双向关系
- ❑ B. 这个bean只能与会话bean建立关系
- ❑ C. 关系只能是一对一或一对多关系
- ❑ D. 获取方法返回类型可以是java.util.List

**2** 对于实体bean提供者使用容器托管持久存储来创建持久关系，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 这个bean必须有远程接口来与另一个实体bean建立双向关系
- ❑ B. 这个bean只能与消息驱动bean建立关系
- ❑ C. 获取方法返回类型可以是java.util.Map
- ❑ D. 关系可以是一对一、一对多或多对多关系

**3** 一个远程客户在一个使用容器托管持久存储的实体bean上调用方法，而这个bean已经被删除，此时会抛出什么异常？

- ❑ A. **`javax.ejb.AccessLocalException`**
- ❑ B. **`javax.ejb.ObjectNotFoundException`**
- ❑ C. **`java.rmi.NoSuchObjectException`**
- ❑ D. **`java.rmi.StubNotFoundException`**
- ❑ E. **`javax.ejb.NoSuchEntityException`**

**4** 给定以下容器托管单向关系：

Foo (0-1) -> Bar (0-1)

以及对象关系：

f1 -> b1

f2 -> b2

运行下面的代码后，哪种情况是正确的？（选出所有正确的答案。）

```
f2.setBar(f1.getBar());
```

- ❑ A. **f1.getBar() == null**
- ❑ B. **b2.getFoo() == null**
- ❑ C. **b1.getFoo() == null**
- ❑ D. 以上都不对

**5** 如果bean A已经从与bean B的关系中删除，在哪些情况下bean A对bean B的存取方法会返回一个非空值？（选出所有正确的答案。）

- ❑ A. 一对一
- ❑ B. 多对一
- ❑ C. 多对多
- ❑ D. 以上都对
- ❑ E. 以上都不对

**6** 哪个部署描述文件元素值的类型必须是集合？（选出所有正确的答案。）

- ❑ A. **<ejb-name>**
- ❑ B. **<cmr-field>**
- ❑ C. **<cmr-field-type>**
- ❑ D. **<ejb-relation>**

**7** 哪个部署描述文件元素必须有另一个部署描述文件元素的两个声明？（选出所有正确的答案。）

- ❑ A. **<ejb-name>**
- ❑ B. **<cmr-field>**
- ❑ C. **<cmr-field-type>**
- ❑ D. **<ejb-relation>**
- ❑ E. **<ejb-relationship-role>**

**8** ejb-relationship-role元素中必须有以下哪一组元素？（选出所有正确的答案。）

- ❑ A. **<cmr-field>, <multiplicity>**
- ❑ B. **<cmr-field>, <relationship-role-source>**
- ❑ C. **<multiplicity>, <relationship-role-name>**
- ❑ D. **<multiplicity>, <relationship-role-source>**
- ❑ E. **<relationship-role-name>, <relationship-role-source>**

**9** 哪些是合法的cmr-field-type元素值？（选出所有正确的答案。）

- ❑ A. String
- ❑ B. Integer
- ❑ C. **java.util.Set**
- ❑ D. **java.util.List**
- ❑ E. **java.util.Collection**

**10** ejb-relation元素中必须有以下哪一组元素？

- ❑ A. **<ejb-relation-name>, <ejb-relationship-role>**
- ❑ B. **<ejb-relationship-role>, <ejb-relationship-role>**
- ❑ C. **<description>, <ejb-relation-name>, <ejb-relationship-role>**
- ❑ D. **<ejb-relation-name>, <ejb-relationship-role>, <ejb-relationship-role>**

**11** 给定CMP bean：CustomerBean、OrderBean和LineItemsBean，它们有以下关系：

CustomerBean (1) <—> OrderBean (n)

OrderBean (1) <—> LineItemsBean (n)

并有以下EJB QL查询：

SELECT DISTINCT OBJECT (c)

FROM Customer c, IN (c.Order) o, IN (o.lineItems) li

WHERE `li.product_type = 'refrigerator'`

以下哪种说法正确地描述了查询的结果？（选出所有正确的答案。）

- ❑ A. 这个查询是非法的
- ❑ B. 所有包括一个冰箱商品的订单
- ❑ C. 所有冰箱商品
- ❑ D. 所有下了冰箱订单的顾客

**12** 给定CMP bean：CustomerBean、OrderBean和LineItemsBean，它们有以下关系：

CustomerBean (1) <—> OrderBean (n)

OrderBean (1) <—> LineItemsBean (n)

下面哪一个查询会返回所有有商品的订单？（选出所有正确的答案。）

- ❑ A. SELECT DISTINCT o
  FROM Order o, IN (o.lineItems) li
- ❑ B. SELECT DISTINCT OBJECT(o)
  FROM Order o, IN (o.lineItems) li
- ❑ C. SELECT OBJECT(o)
  FROM Order o
  WHERE o.lineItems = 0
- ❑ D. SELECT OBJECT(o)
  FROM Order o

**13** 关于EJB QL路径表达式，以下哪一种说法是正确的？（选出所有正确的答案。）

- ❑ A. 在路径表达式中，(.)被认为是导航操作符
- ❑ B. 路径表达式可以用`cmr-field`或`cmp-field`结尾
- ❑ C. 用`cmr-field`结尾的路径表达式不能进一步用点记法导航
- ❑ D. 路径表达式可以用一个单值结束，也可以用一个集合值结束

**14** 关于EJB QL查询，以下哪一种说法是正确的？（选出所有正确的答案。）

- ❑ A. SELECT、FROM和WHERE这3种子句中，仅SELECT子句是必要的
- ❑ B. SELECT指定了查询域
- ❑ C. WHERE子句确定了所选对象的类型
- ❑ D. EJB QL查询可以有参数

**15** 关于EJB QL WHERE子句，以下哪一种说法是正确的？（选出所有正确的答案。）

- ❑ A. WHERE子句中用的标识变量只能在FROM子句中定义
- ❑ B. 标识变量可以表示单值，也可以表示一个集合
- ❑ C. 输入参数的个数必须等于查找方法或选择方法的参数个数
- ❑ D. 输入子句只能用于WHERE子句

**16** 给定以下EJB QL表达式：

```
p.discount BETWEEN 10 AND 15
```

哪个表达式与之等价？

- ❑ A. `p.discount > 10 AND p.discount < 15`
- ❑ B. `p.discount >= 10 AND p.discount < 15`
- ❑ C. `p.discount > 10 and p.discount <= 15`
- ❑ D. `p.discount >= 10 and p.discount <= 15`

**17** 关于EJB QL IN表达式，以下哪一种说法是正确的？（选出所有正确的答案。）

- ❑ A. **single_valued_path_expression** 的值必须是String
- ❑ B. NOT逻辑操作符可以用于IN表达式中
- ❑ C. 串直接量列表可以为空
- ❑ D. 以下表达式是合法的：**o.country** IN（“UK”，“US”）

**18** 给定以下EJB QL表达式：

product.code LIKE ‘F_s’

以下哪个值的比较结果为true？（选出所有正确的答案。）

- ❑ A. Fs
- ❑ B. F1s
- ❑ C. FXs
- ❑ D. F37s
- ❑ E. Fits

# COFFEE CRAM
# 模拟测验答案

**1** 对于实体bean提供者使用容器托管持久存储来创建持久关系，以下哪些说法是正确的？（选出所有正确的答案。）

- ☑ A. 这个bean必须有本地接口来与另一个实体bean建立双向关系
- ❑ B. 这个bean只能与会话bean建立关系
- ❑ C. 关系只能是一对一或一对多关系
- ❑ D. 获取方法返回类型可以是java.util.List — 必须是Collection或Set

**2** 对于实体bean提供者使用容器托管持久存储来创建持久关系，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 这个bean必须有远程接口来与另一个实体bean建立双向关系 — 必须是本地接口
- ❑ B. 这个bean只能与消息驱动bean建立关系
- ❑ C. 获取方法返回类型可以是java.util.Map — 只能是Collection或Set
- ☑ D. 关系可以是一对一、一对多或多对多关系

**3** 一个远程客户在一个使用容器托管持久存储的实体bean上调用方法，而这个bean已经被删除，此时会抛出什么异常？ （规范：(32)）

- ❑ A. `javax.ejb.AccessLocalException`
- ❑ B. `javax.ejb.ObjectNotFoundException`
- ☑ C. `java.rmi.NoSuchObjectException`
- ❑ D. `java.rmi.StubNotFoundException`
- ❑ E. `javax.ejb.NoSuchEntityException`

**4** 给定以下容器托管单向关系： （规范：(38)

Foo (0-1) -> Bar (0-1)

以及对象关系：

f1 -> b1

f2 -> b2

运行下面的代码后，哪种情况是正确的？（选出所有正确的答案。）

```
f2.setBar(f1.getBar());
```

- ☑ A. `f1.getBar() == null`
- ☑ B. `b2.getFoo() == null`
- ☐ C. `b1.getFoo() == null`
- ☐ D. 以上都不对

setBar()方法"打断了"现有的对象关系，并创建了一个新的关系

**5** 如果bean A已经从与bean B的关系中删除，在哪些情况下bean A对bean B的存取方法会返回一个非空值？（选出所有正确的答案。） （规范：(32)

- ☐ A. 一对一
- ☐ B. 多对一
- ☑ C. 多对多
- ☐ D. 以上都对
- ☐ E. 以上都不对

**6** 哪个部署描述文件元素值的类型必须是集合？（选出所有正确的答案。） （规范：463～464)

- ☐ A. `<ejb-name>`
- ☐ B. `<cmr-field>`
- ☑ C. `<cmr-field-type>` — Collection或Set
- ☐ D. `<ejb-relation>`

**7** 哪个部署描述文件元素必须有另一个部署描述文件元素的两个声明？（选出所有正确的答案。） （规范：467～468）

- ❑ A. **<ejb-name>**
- ❑ B. **<cmr-field>**
- ❑ C. **<cmr-field-type>**
- ☑ D. **<ejb-relation>** 它要有两个<ejb-relationship-role>子元素
- ❑ E. **<ejb-relationship-role>**

**8** ejb-relationship-role元素中必须有以下哪一组元素？（选出所有正确的答案。） （规范：468）

- ❑ A. **<cmr-field>, <multiplicity>**
- ❑ B. **<cmr-field>, <relationship-role-source>**
- ❑ C. **<multiplicity>, <relationship-role-name>**
- ☑ D. **<multiplicity>, <relationship-role-source>** 关系里有谁，它们如何相关
- ❑ E. **<relationship-role-name>, <relationship-role-source>**

**9** 哪些是合法的cmr-field-type元素值？（选出所有正确的答案。） （规范：464）

- ❑ A. String
- ❑ B. Integer
- ☑ C. **java.util.Set**
- ❑ D. **java.util.List**
- ☑ E. **java.util.Collection**

可以是本地接口类型，或者是本地接口类型的Collection或Set

**10** ejb-relation元素中必须有以下哪一组元素？ （规范：467～468）

- ❑ A. **<ejb-relation-name>, <ejb-relationship-role>**
- ☑ B. **<ejb-relationship-role>, <ejb-relationship-role>** 要有关系双方
- ❑ C. **<description>, <ejb-relation-name>, <ejb-relationship-role>**
- ❑ D. **<ejb-relation-name>, <ejb-relationship-role>, <ejb-relationship-role>**

**11** 给定CMP bean：CustomerBean、OrderBean和LineItemsBean，它们有以下关系：

(规范：233)

CustomerBean (1) <—> OrderBean (n)

OrderBean (1) <—> LineItemsBean (n)

并有以下EJB QL查询：

SELECT DISTINCT OBJECT (c)

FROM Customer c, IN (c.Order) o, IN (o.lineItems) li

WHERE **li.product_type = 'refrigerator'**

以下哪种说法正确地描述了查询的结果？（选出所有正确的答案。）

- ❑ A. 这个查询是非法的
- ❑ B. 所有包括一个冰箱商品的订单
- ❑ C. 所有冰箱商品
- ☑ D. 所有下了冰箱订单的顾客

SELECT子句指定了这个查询必须返回Customer对象

**12** 给定CMP bean：CustomerBean、OrderBean和LineItemsBean，它们有以下关系：

(规范：236)

CustomerBean (1) <—> OrderBean (n)

OrderBean (1) <—> LineItemsBean (n)

下面哪一个查询会返回所有有商品的订单？（选出所有正确的答案。）

- ❑ A. SELECT DISTINCT o
  FROM Order o, IN (o.lineItems) li
- ☑ B. SELECT DISTINCT OBJECT(o)
  FROM Order o, IN (o.lineItems) li
- ❑ C. SELECT OBJECT(o)
  FROM Order o
  WHERE o.lineItems = 0
- ☑ D. SELECT OBJECT(o)
  FROM Order o

**13** 关于EJB QL路径表达式，以下哪一种说法是正确的？（选出所有正确的答案。）

（规范：225～226）

- ☑ A. 在路径表达式中，**(.)**被认为是导航操作符
- ☑ B. 路径表达式可以用**cmr-field**或**cmp-field**结尾
- ☐ C. 用 **cmr-field** 结尾的路径表达式不能进一步用点记法导航
- ☑ D. 路径表达式可以用一个单值结束，也可以用一个集合值结束

**14** 关于EJB QL查询，以下哪一种说法是正确的？（选出所有正确的答案。）

（规范：218～219）

- ☐ A. SELECT、FROM和WHERE这3种子句中，仅SELECT子句是必要的
- ☐ B. SELECT指定了查询域
- ☐ C. WHERE子句确定了所选对象的类型
- ☑ D. EJB QL查询可以有参数

**15** 关于EJB QL WHERE子句，以下哪一种说法是正确的？（选出所有正确的答案。）

（规范：227～228）

- ☑ A. WHERE子句中用的标识变量只能在FROM子句中定义
- ☐ B. 标识变量可以表示单值，也可以表示一个集合
- ☐ C. 输入参数的个数必须等于查找方法或选择方法的参数个数
- ☑ D. 输入子句只能用于WHERE子句

**16** 给定以下EJB QL表达式：

（规范：229）

```
p.discount BETWEEN 10 AND 15
```

哪个表达式与之等价？

- ☐ A. `p.discount > 10 AND p.discount < 15`
- ☐ B. `p.discount >= 10 AND p.discount < 15`
- ☐ C. `p.discount > 10 and p.discount <= 15`
- ☑ D. `p.discount >= 10 and p.discount <= 15`

**17** 关于EJB QL IN表达式，以下哪一种说法是正确的？（选出所有正确的答案。） （规范：229～230）

☑ A. **single_valued_path_expression** 的值必须是String

☑ B. NOT逻辑操作符可以用于IN表达式中

☐ C. 串直接量列表可以为空

☐ D. 以下表达式是合法的：**o.country** IN（“UK”，“US”）

当心双引号！

（规范：230）

**18** 给定以下EJB QL表达式：

product.code LIKE ‘F_s’

以下哪个值的比较结果为true？（选出所有正确的答案。）

☐ A. Fs

☑ B. F1s

☑ C. FXs

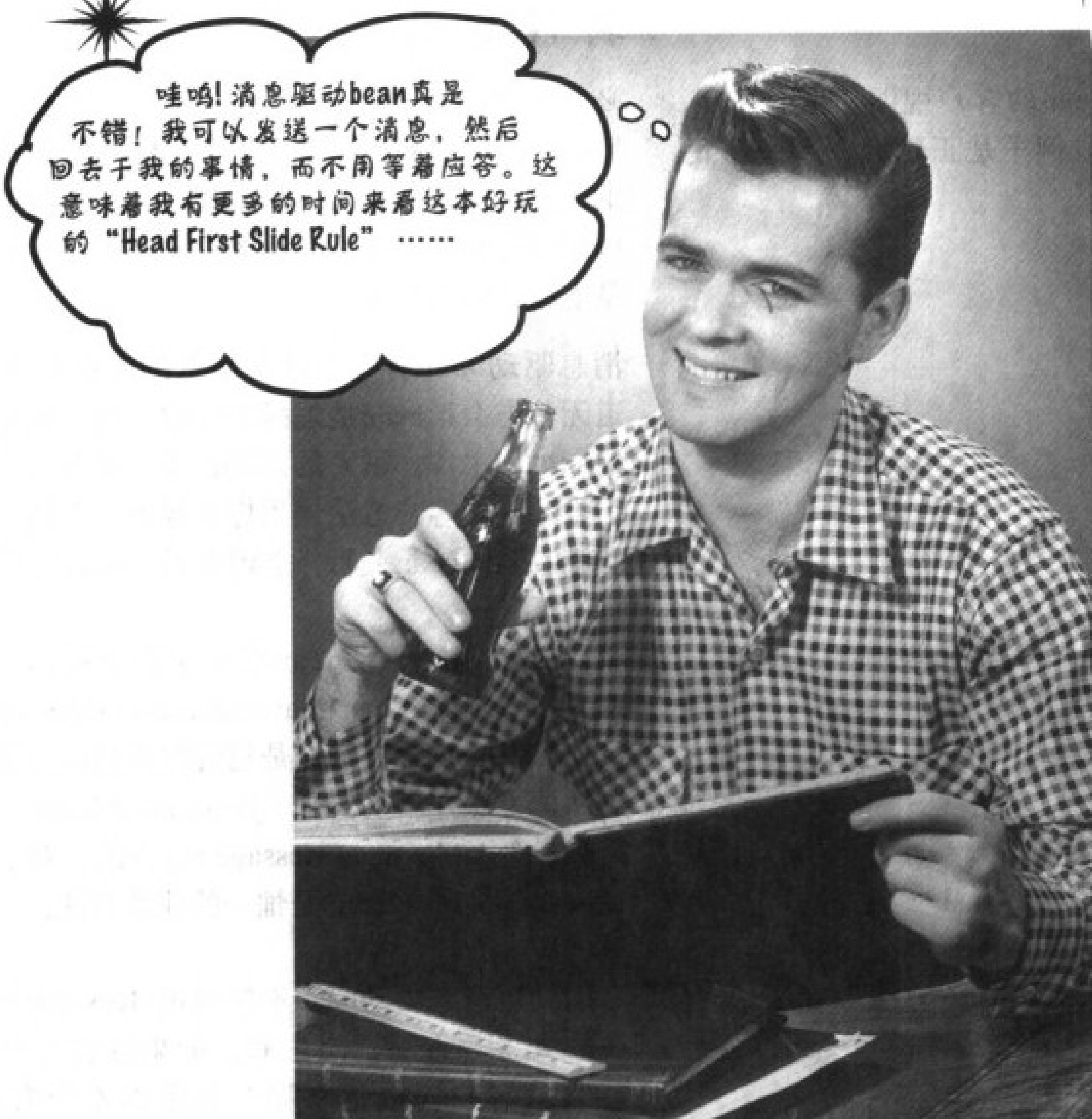
☐ D. F37s

☐ E. Fits

%字符是一个字符序列的通配符，_是单个字符的通配符

# 8 消息驱动bean

## 得到消息




**接收消息很有意思。**虽然不是特别地好玩，比如说，比不上收到EBay给你的一个真正的Smurf™ 车灯包裹，但确实很有意思，也很高效。假设你向EBay发了一个订单，在包裹递送到之前你就不能离开家（否则就会错过）。会话bean和实体bean就是这样，因为对bean客户接口的所有调用都是同步的（无论是home接口、组件接口、本地接口还是远程接口，都一样）。但是利用消息驱动bean，客户可以建立一个消息，把它发给一个消息服务。然后客户可以走开。这个消息服务则会把消息发送给容器，容器再把消息交给一个消息驱动bean。

# 消息驱动bean

| 标准大纲： | | 具体含义： |
|---|---|---|
| 10.1 | 关于消息驱动bean的客户视图和生命周期，区别哪些说法或例子是正确的，哪些不正确。 | 必须知道消息驱动bean没有客户！这意味着，没有home接口，没有组件接口，而且也不存在本地和远程的问题。当然消息驱动bean确实有一个调用者，但这个调用者是容器。我们不认为容器是一个客户。容器应该是老板，而不是顾客。<br><br>消息驱动bean有一个非常简单的生命周期，如果你知道无状态会话bean是怎样工作的，也就知道了消息驱动bean如何工作。惟一的差别在于，并非将bean从池中取出来为一个客户方法调用提供服务，而是容器从池中取出消息驱动bean来为一个到来的JMS消息提供服务。 |
| 10.2 | 明确消息驱动bean必须实现哪些接口和方法。 | 必须知道消息驱动bean实现了两个接口：javax.ejb.MessageDrivenBean和javax.jms.MessageListener。javax.ejb.MessageDrivenBean是包含容器回调方法的接口（容器回调方法并不多！），javax.jms.MessageListener接口只有一个onMessage(Message m) 方法。对于一个消息驱动bean，onMessage()是惟一的业务方法。 |
| 10.3 | 明确MessageDrivenContext接口方法如何使用，并了解这些方法的行为。 | 必须知道消息驱动bean不能调用MessageDrivenContext中的大多数方法。想想看。如果没有客户，你又怎么能得到客户安全信息呢？如果根本没有客户，这意味着没有客户视图，那怎么能调用getEJBObject() 或getEJBHome()呢？你根本没有home！而且也没有EJBObject。 |
| 10.4 | 对于一个消息驱动bean，给定一个列表，区别哪些是bean提供者的职责，哪些是容器提供者的职责。 | 你必须认识到，如果没有客户就不能声明任何受查异常。否则，谁来捕获这些异常呢？而且，与无状态会话bean一样，如果在一个方法中开始一个事务（这只能是一个BMT bean在执行onMessage()），就必须在方法完成前结束这个事务。 |

## 想像一下这种情况……

你必须让某个人完成一件非常重要的任务。

但并不知道这个人要花多长时间才能完成任务。

你必须待在原地等待，直到任务完成为止。

在等待的过程中，你无法做任何其他事情。




真是的……能有
多难呀，要这么久？
服务器到底在干什么
呢？




客户

有消息驱动bean之前



## 真糟糕，它们不是消息驱动bean

方法调用（不论是本地还是远程）都是同步的。调用者要一直等待，直到服务器返回。

消息驱动bean（EJB 2.0中新增的）则在客户（消息发送者）和服务器（消息接收者）之间提供了一种异步通信。

按消息传递的术语，发送者称为消息生产者（message Producer），接收者称为消息消费者（message Consumer）。

利用消息传递，生产者发送一个消息，然后回去做他自己的事情。他甚至不会等着消费者接收到消息，更不用说等着消费者处理完。

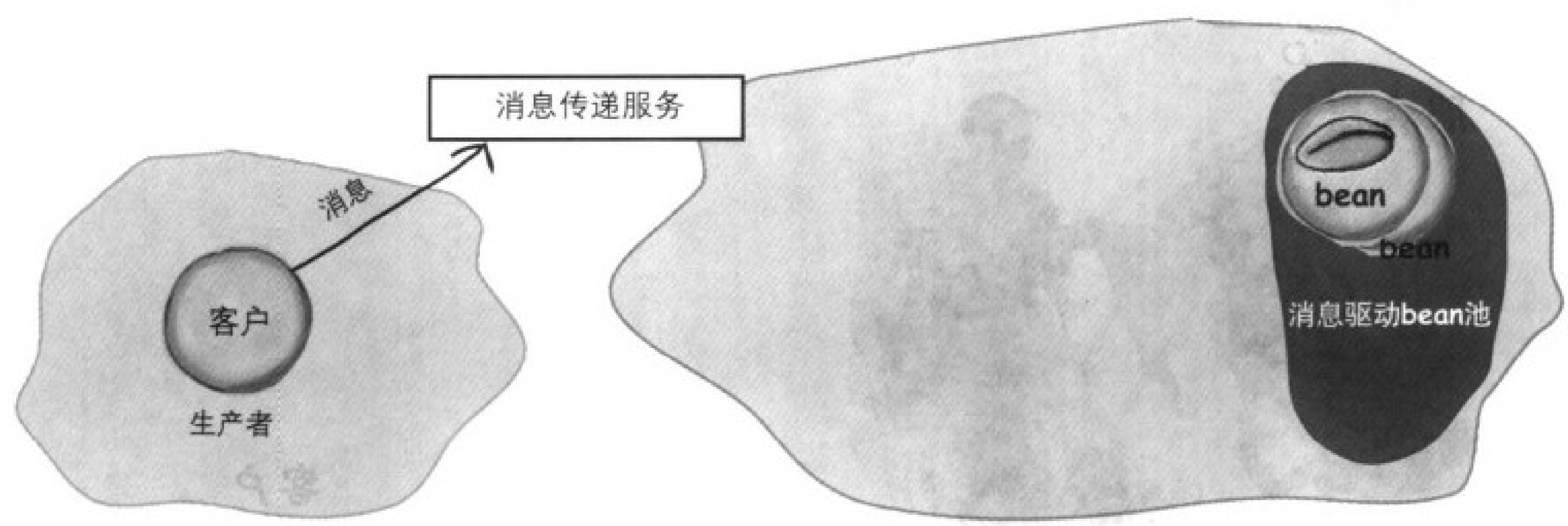
消费者得到一个消息时，他会进行处理，与此同时，客户仍能有自己的生活。

有了消息驱动bean之后

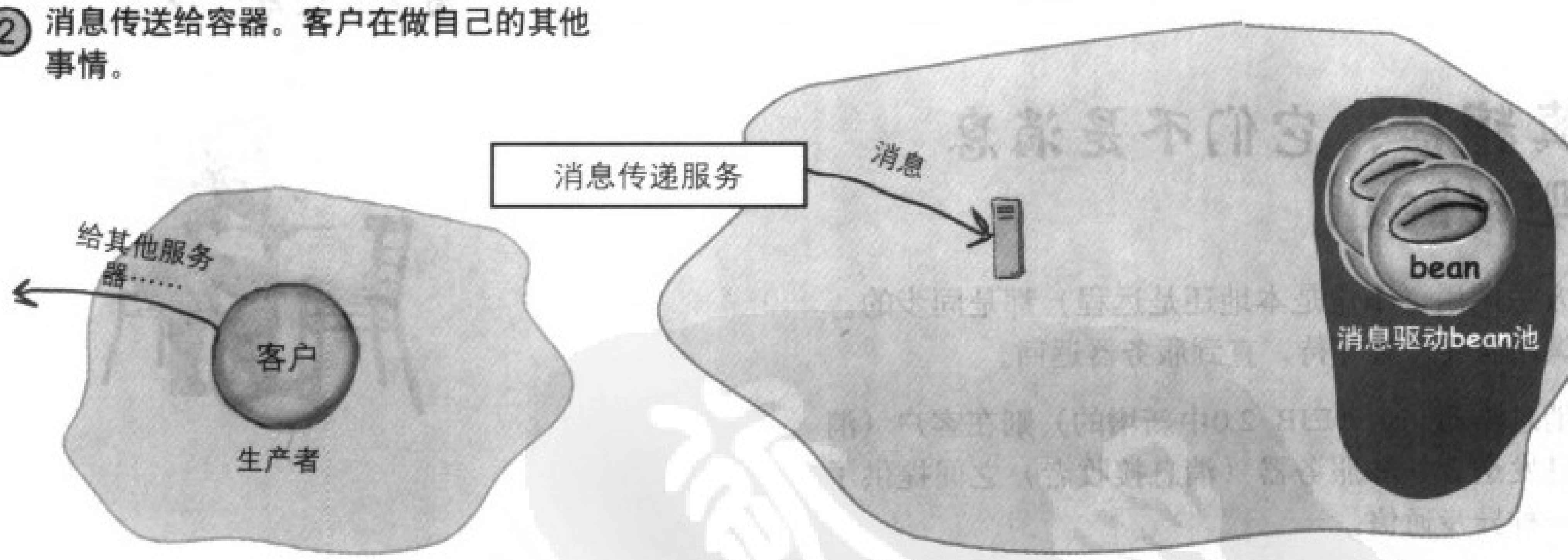


## 消息驱动bean概述

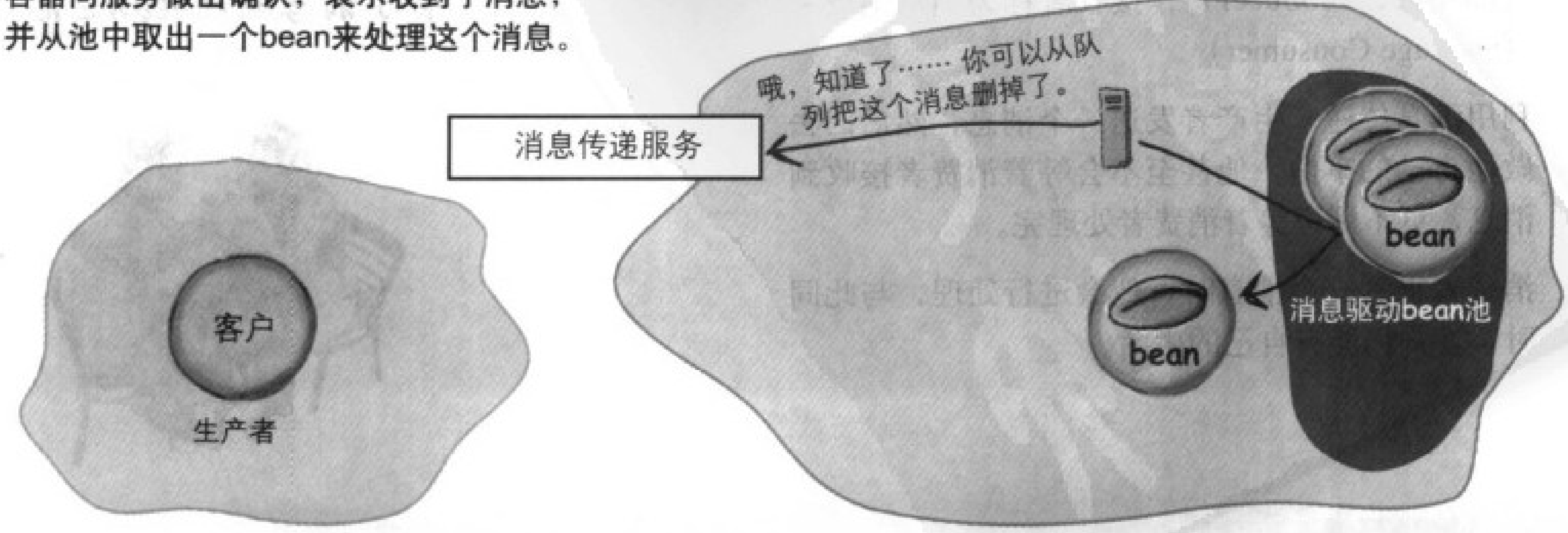
① 客户（生产者）向消息传递服务发送一个消息。



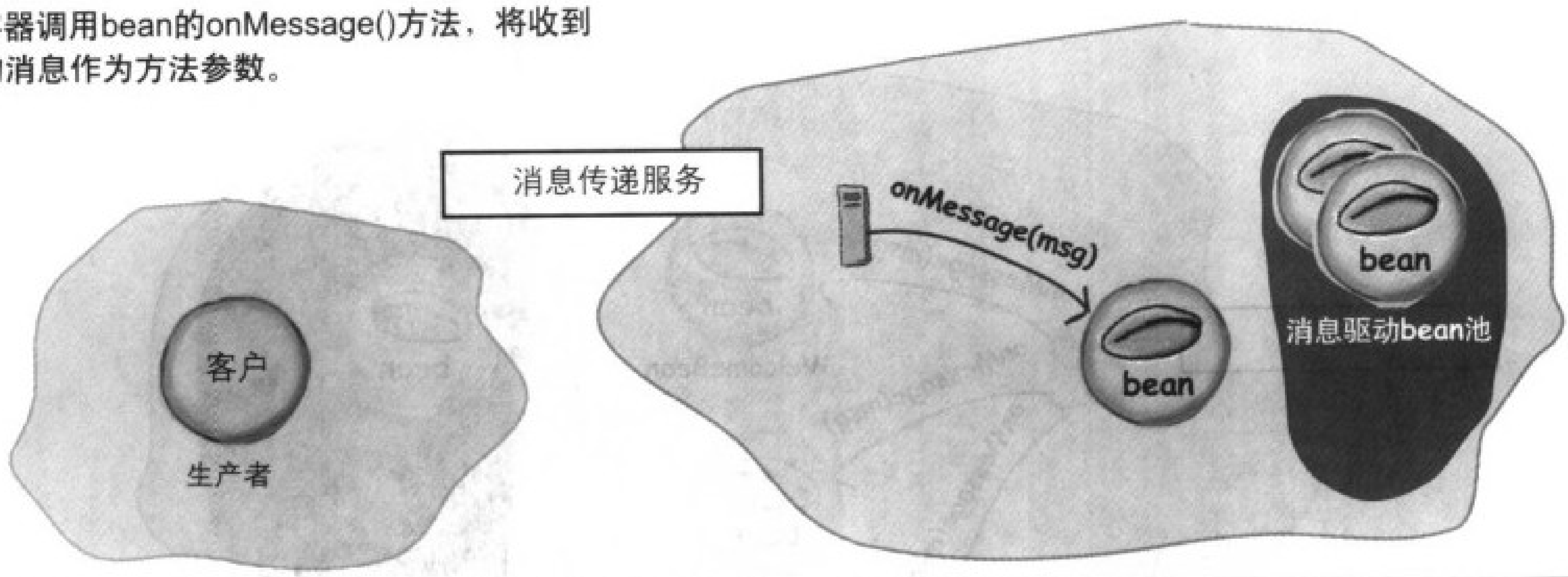

② 消息传送给容器。客户在做自己的其他事情。




③ 容器向服务做出确认，表示收到了消息，并从池中取出一个bean来处理这个消息。

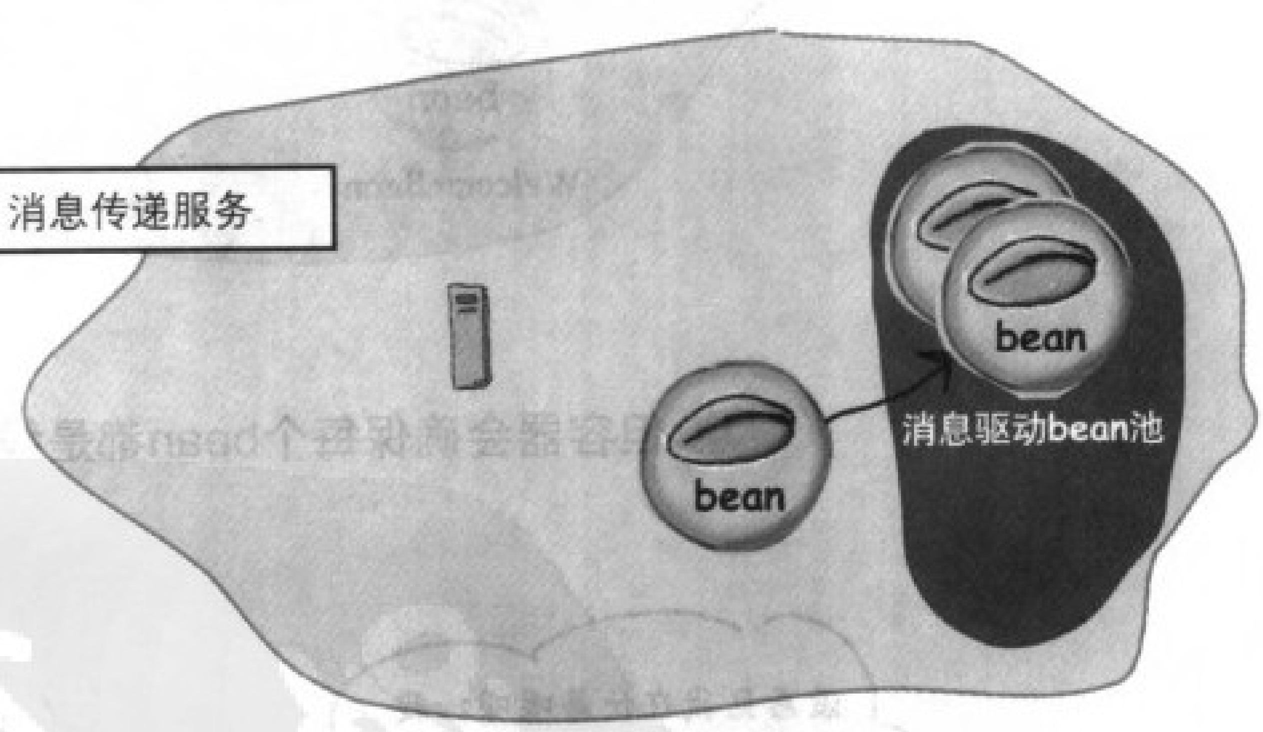


④ 容器调用bean的onMessage()方法，将收到的消息作为方法参数。




⑤ bean的事务提交，容器把bean发回到池中。

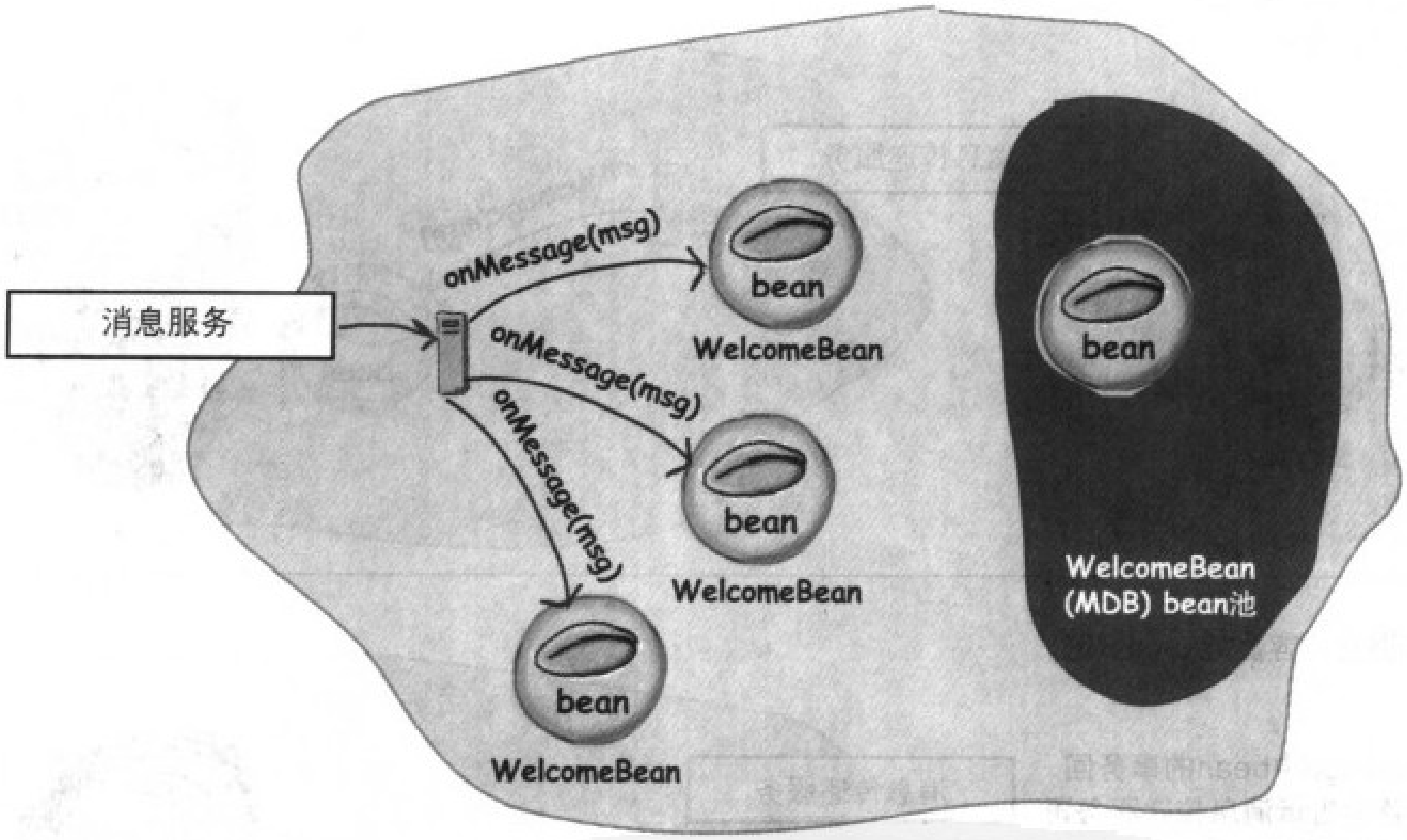
如果在onMessage中bean的事务回滚了，容器就会告诉消息传递服务再把消息放回到队列中。




### 看看这是不是和无状态会话bean很像呢？

类似于无状态会话bean，消息驱动bean对于客户没有惟一的身分（更确切地讲，它们根本没有客户，因为一般不认为容器是bean的一个客户），消息驱动bean会放到池里，而且它们各自没有会影响其业务逻辑的状态。这说明，一个（某种特定类型）消息驱动bean与来自同一个home的其他bean是相同的。这与无状态bean完全一样。

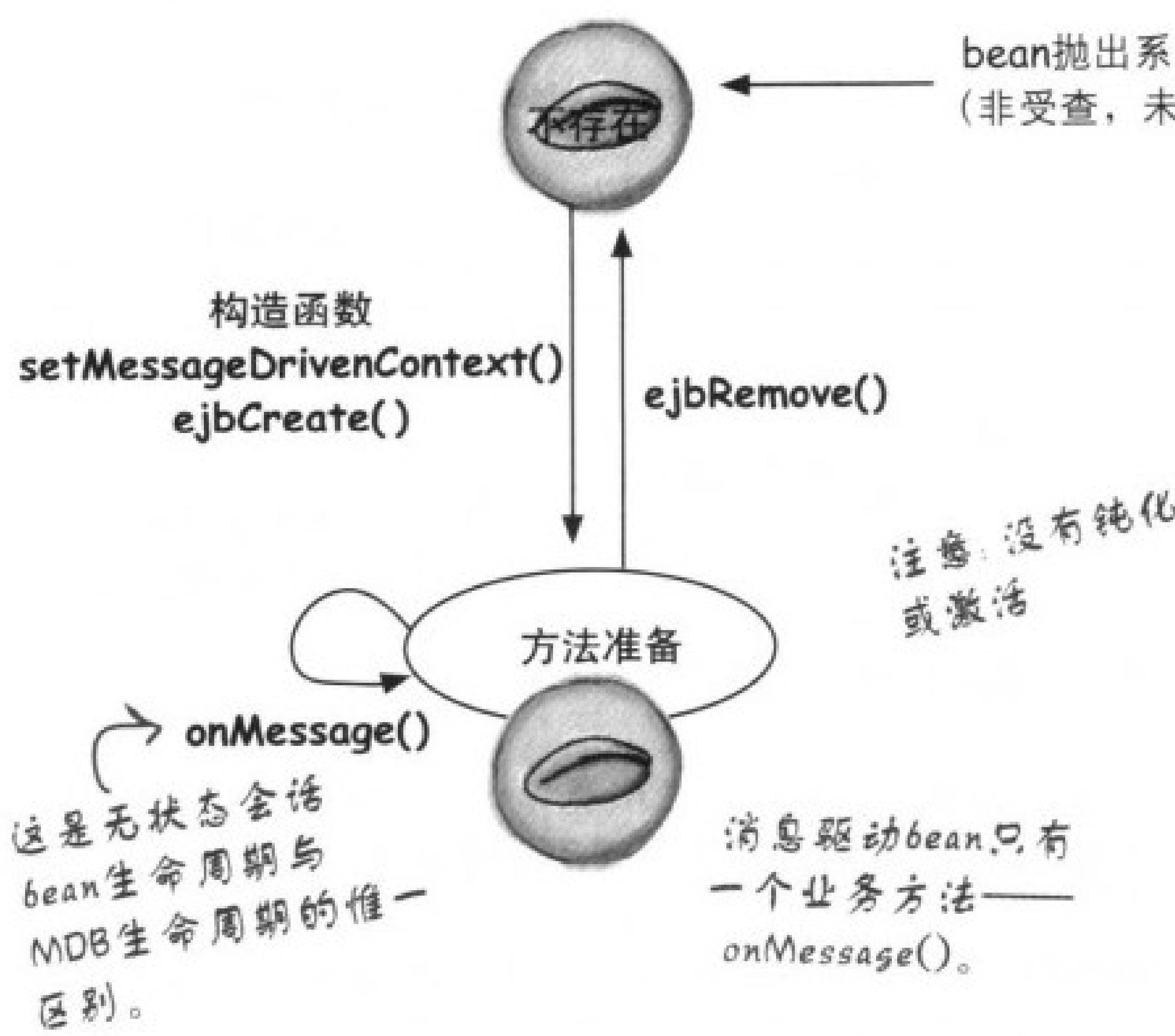
同类型的多个bean可以并发地处理消息。




但容器会确保每个bean都是线程安全的！



## 消息驱动bean的生命周期与无状态会话bean的生命周期很相似




```
public void onMessage(Message msg) {
    // 打开消息，对它做些处理

}
```

就是这样！这是消息驱动bean惟一需要做的业务逻辑，或者说这也是它惟一能做的业务逻辑（除了用于创建、删除和设置上下文的3个容器回调）。

# 消息驱动bean类

```
package headfirst;

import javax.ejb.*;
import javax.jms.*;

public class WelcomeBean implements MessageDrivenBean, MessageListener {

    private MessageDrivenContext context;

    public void ejbCreate() { }

    public void ejbRemove() { }

    public void setMessageDrivenContext(MessageDrivenContext ctx) {
        context = ctx;
    }

    public void onMessage(Message msg) {
        // 处理消息
       try {

         if (msg instanceof TextMessage) {
          TextMessage message = (TextMessage) msg;

          System.out.println(message.getText());
         }
       } catch (JMSException ex) {
          ex.printStackTrace();
       }
    }
}
```

这是以前没见过的一个导入语句：导入javax.jms包

这是两个要实现的接口！

javax.ejb.MessageDrivenBean里包括容器回调，javax.jms.MessageListener里有一个onMessage()回调方法。

就像SessionContext和EntityContext一样，MessageDrivenContext也扩展了EJBContext

必须有一个无参数的ejbCreate()

与无状态会话bean的工作类似……当容器想缩小池的大小时就会调用这个方法

和以往一样……保存你的上下文

终于！这是真正的业务方法，这也是MessageListener接口中定义的惟一的一个方法。

TextMessage是一个Message，不过还可以是另外某种消息，如BytesMessage、MapMessage、ObjectMessage或StreamMessage。所以在类型强制转换之前要先用instanceof测试。

注意，这是一个受查异常！它来自getText()方法。

# 编写消息驱动bean：bean提供者的任务

这一回没有客户视图，所以我不用与home接口或组件接口中的方法匹配。其实根本就不存在home接口或组件接口。



bean类中要放3类方法：

① bean法则：ejbCreate()方法

在bean中要编写一个无参数的ejbCreate()方法。它不与任何接口中的方法匹配，也并非来自哪个接口。之所以有这个方法，只是因为必须如此。




② MessageListener实现：onMessage()

这是你的业务方法。这也是惟一的业务方法。




③ MessageDrivenBean实现：容器回调

实现MessageDrivenBean接口的两个方法，你的bean必须以正式的Java方式实现这个接口（也就是说，在bean类中或者它的某个超类中使用“implements MessageDrivenBean”声明）。

**Sharpen your pencil**

在bean中放的3种方法中，将编译器会检查的方法打勾。

**编译器检查吗？**

- ❑ ejbCreate()
- ❑ onMessage()
- ❑ ejbRemove()和setMessageDrivenContext()

## 消息驱动bean类的规则

1. bean类必须实现javax.ejb.MessageDrivenBean和javax.jms.MessageListener接口。

```
public class WelcomeBean implements MessageDrivenBean, MessageListener {
```

2. bean类必须是公共的（public），但不能是最终类（final），而且不能是抽象类（abstract）。

3. bean类必须有一个公共的无参数构造函数（这与其他类型的bean一样……这样容器才能建立一个新实例）。

```
public WelcomeBean() { }
```

尽可能就用编译器建立的默认构造函数。不过，如果你确实有一个构造函数，一定要确保它是无参数的，而且其中不要放任何代码。要等ejbCreate()完成了才能做工作。

4. bean类必须有一个无参数的ejbCreate()方法。它必须是公共的，但不能是最终（final），也不能是静态方法（static），返回类型必须是void。

```
public void ejbCreate() { }
```

在这里放初始化代码。等调用这个方法的时候，你已经有了你的上下文了。

5. bean类必须定义MessageListener接口中的onMessage()方法。这个方法必须是公共的，但不能是最终（final），也不能是静态方法（static），返回类型必须是void，而且这个方法只取一个类型为javax.jms.Message的参数。

```
public void onMessage(Message msg) { ... }
```

真正的业务方法。你的逻辑都放在这里……

6. 必须有MessageDrivenBean的ejbRemove()和setMessageDrivenContext()方法，并且要严格遵循接口中的声明。

```
public void ejbRemove() { }
public void setMessageDrivenContext(MessageDrivenContext ctx) {
   context = ctx;
}
```

7. 这个类中所有方法都不允许抛出应用异常（而且也不能声明运行时异常，尽管从理论上讲这是合法的）。

**消息驱动bean不能声明受查异常！！**

在第10章你还会了解更多有关异常的内容，不过现在可以简单想想看。这里没有客户！那么你觉得谁会用一个try/catch来包住这些调用呢？如果你尝试这样做，容器会笑话你的。你必须捕获和处理收到的所有受查异常！！

# 注意到代码中少了什么吗？

## 我们没有说监听的是哪种类型的消息，也没有说这些消息可能来自哪里。

在代码中没有放任何东西来指示JMS目标。你可能猜得出来会在什么时候指定……对了，就是在部署时。

不过，作为一个bean提供者，你要告诉部署人员你在从一个主题（topic）查找消息还是从一个队列（queue）查找。为此，我们要在部署描述文件中使用一个新标记，专门用于消息驱动bean。在部署时，部署人员会把你的bean与一个特定的主题或队列绑定，这个主题或队列配置为EJB服务器中的一个资源。

```
<message-driven-destination>
    <destination-type>javax.jms.Topic</destination-type>
</message-driven-destination>
```

你告诉部署人员希望消息从一个主题（Topic）来，还是从一个队列（Queue）来。

## 消息驱动bean的完整的部署描述文件

```
<enterprise-beans>
    <message-driven>
      <ejb-name>WelcomeNewCustomer</ejb-name>
      <home>headfirst.CustomerHome</home>
      <remote>headfirst.Customer</remote>

      <ejb-class>headfirst.WelcomeBean</ejb-class>
      <transaction-type>Container</transaction-type>
      <message-driven-destination>
         <destination-type>javax.jms.Topic</destination-type>
      </message-driven-destination>
    </message-driven>
</enterprise-beans>
```

这里没有客户视图！没有本地视图，没有远程视图。根本就没有任何视图。所以不用放对应home接口和组件接口的标记。

（后面几页还会介绍一个可选的标记）

# 主题和队列

消息传递有两种类型：主题和队列。不过主题又可能分成两种子类型：持久订购和非持久订购。

队列有些像FIFO列表（不过不能保证遵循先进先出顺序）。生产者发送一个消息，这个消息只面向一个消费者。要是有人处理了这个消息，那肯定就是这个消费者。员工偿付（employee reimbursement）系统就是这样一个例子，员工向消息服务发出他的偿付请求，会计部门的一个人就会处理这个请求。此时不需要把请求发给任何其他人。如果下一步是把偿付请求发送给管理部门处理，会计部门可以向另一个目标发送一个消息，如ManagerApproval队列。

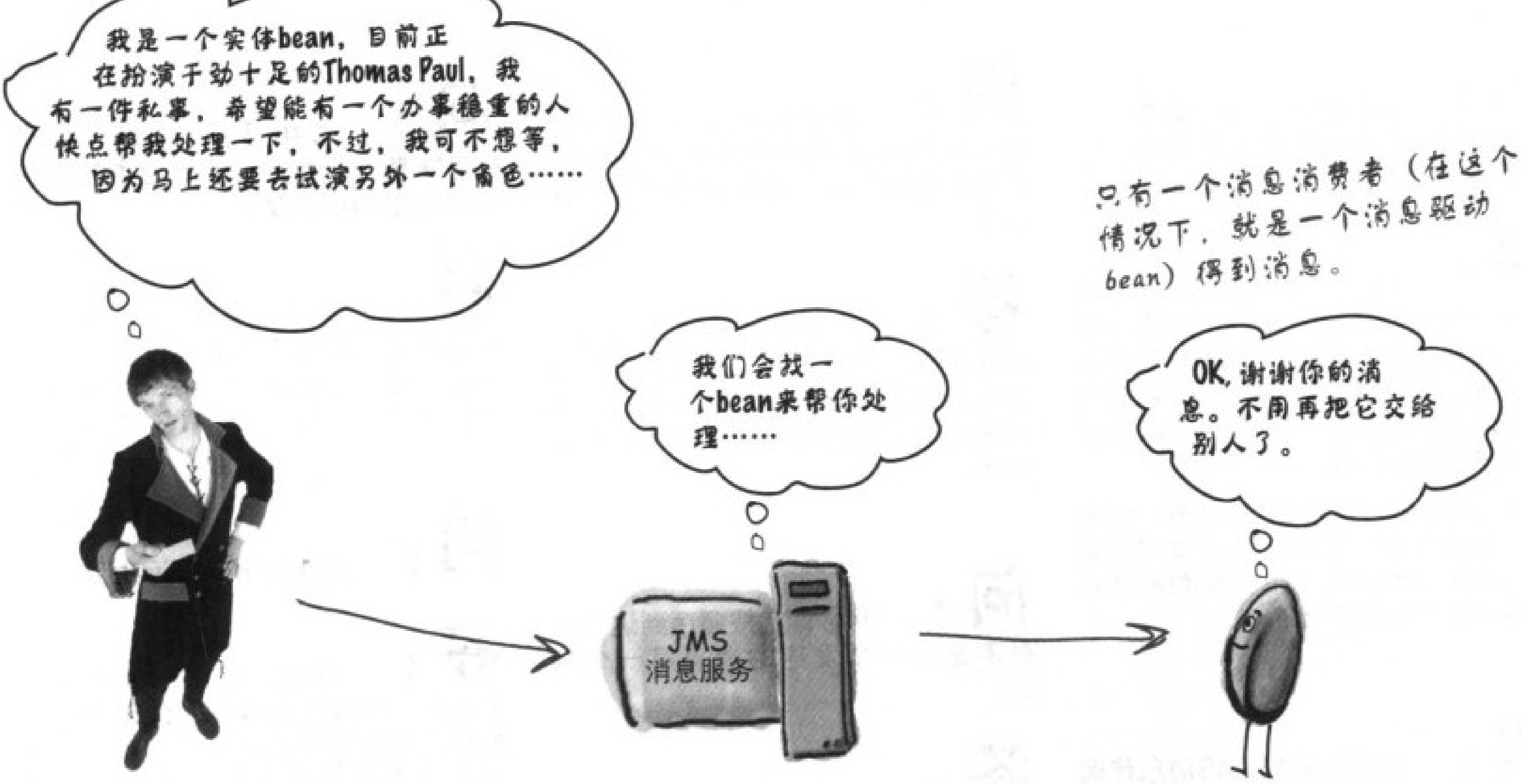
主题使用一种发布和订购模型，生产者发送一个消息，只要是作为消费者监听的任何人都会得到这个消息的一个副本。这个工作就像一个邮件列表。如果有哪个订购者没有得到消息，生产者并不关心。

如果主题订购者想确保能看到所有消息，例如，还包括他离线时所积累的消息，就可以请求一个持久订购。你的选择几乎总是持久订购，因为你往往希望得到所有的消息。想想吧……什么是非持久订购呢？这就像是给你递送杂志的时候你必须在家，否则你就看不到杂志了。

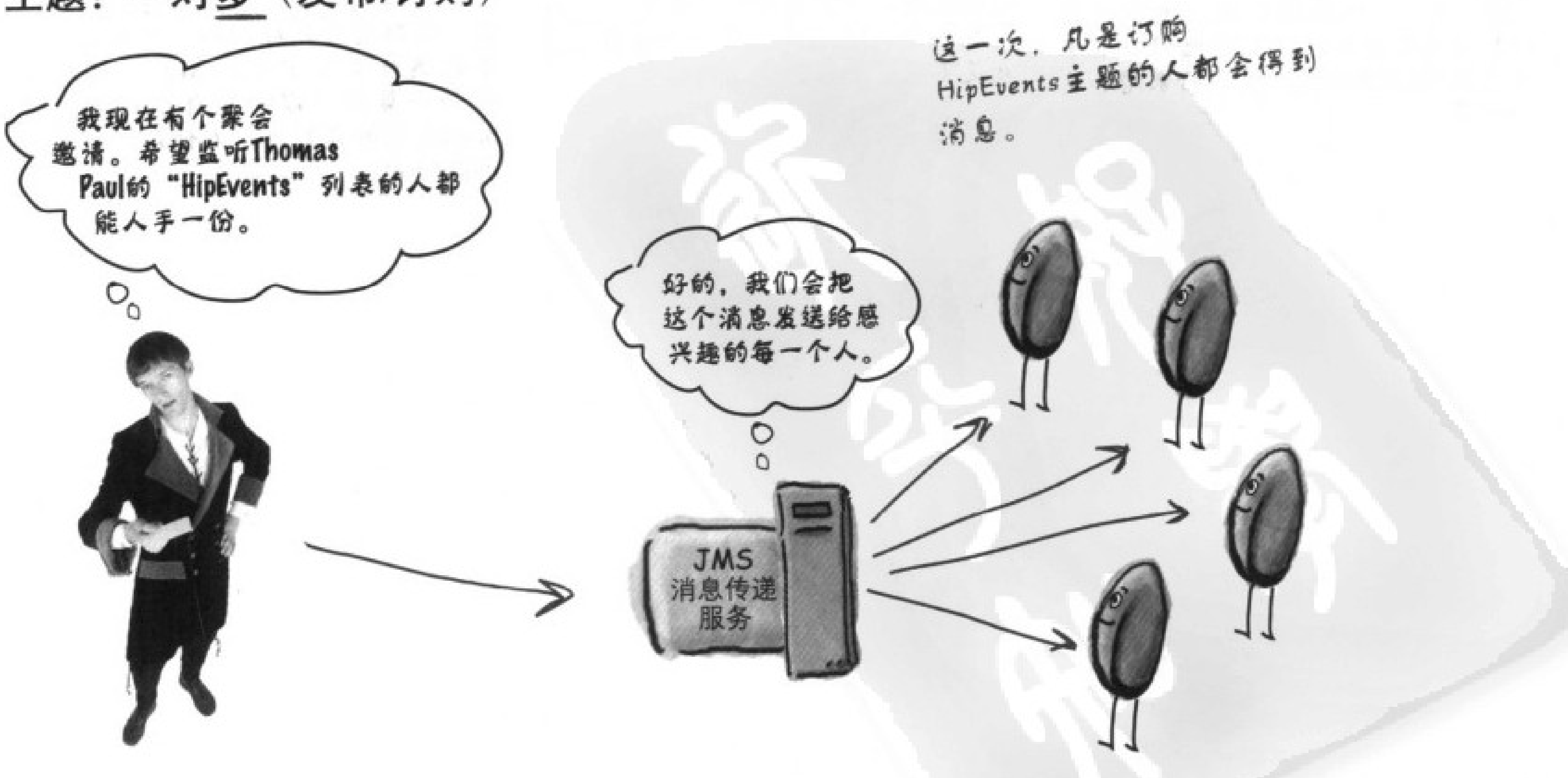
企业系统往往会使用队列或持久主题订购。非持久主题订购有时可能很合适，但是如果我是一个消息消费者，选择非持久订购就相当于在说："能不能得到这些消息，我不是太在乎。"



## 队列：一对一（点对点）




## 主题：一对多（发布/订购）



## there are no Dumb Questions

**问：JMS在这里是怎么用的？另外，如果我有一个消息驱动bean，到哪里得到消息传递服务呢？**

答：JMS 1.02是EJB 2.0要求的Java消息传递服务API。这意味着你的EJB容器必须包含有某种消息传递服务，不过许多开发商还可以处理多种消息传递服务，只要这些服务支持JMS API就可以。

JMS API的工作与JDBC很相似，你有一个驱动程序，它知道怎么拿到你的标准JMS API调用，并把它们翻译成底层消息系统理解的东西。

**问：我可以从非JMS消息传递服务得到消息吗？**

答：现在还不能，也就是在EJB 2.0里还不行，不过他们（名声不太好的J2EE小组）希望将来增加这个功能。

**问：为什么不让MessageDrivenBean扩展MessageListener接口呢？这样的话，就不用实现两个接口了。**

答：看看前面的那个问题。J2EE小组不想把所有消息驱动bean都锁定为JMS监听者。总有一天，对应不同的消息类型可能有多种类型的监听者。

**问：消息肯定按顺序到达吗？是不是第一个发送的消息第一个到达？**

答：不是!!最好不要指望这样，否则就可能发生不好的事情。例如，你可能会在得到“下订单”消息之前先得到“撤销订单”消息。设计系统和编写bean时要有一个假设，这就是不能完全保证消息的顺序。

**问：如果是一个主题，是不是意味着一个特定订购者池中的所有bean都会得到消息？**

答：不是这样的!要记住，一个bean就可以代表所有其他bean。容器只会把消息送给池中的一个bean。除非……不过这一点不用担心。

**问：除非什么？告诉我吧！**

答：好吧，好吧。但考试不考这个，所以大可放心。假设你的服务器正以集群配置运行，在此可能运行着多个服务器实例。在这种情况下，容器如何向消息传递服务表现自己，这一点不能保证。

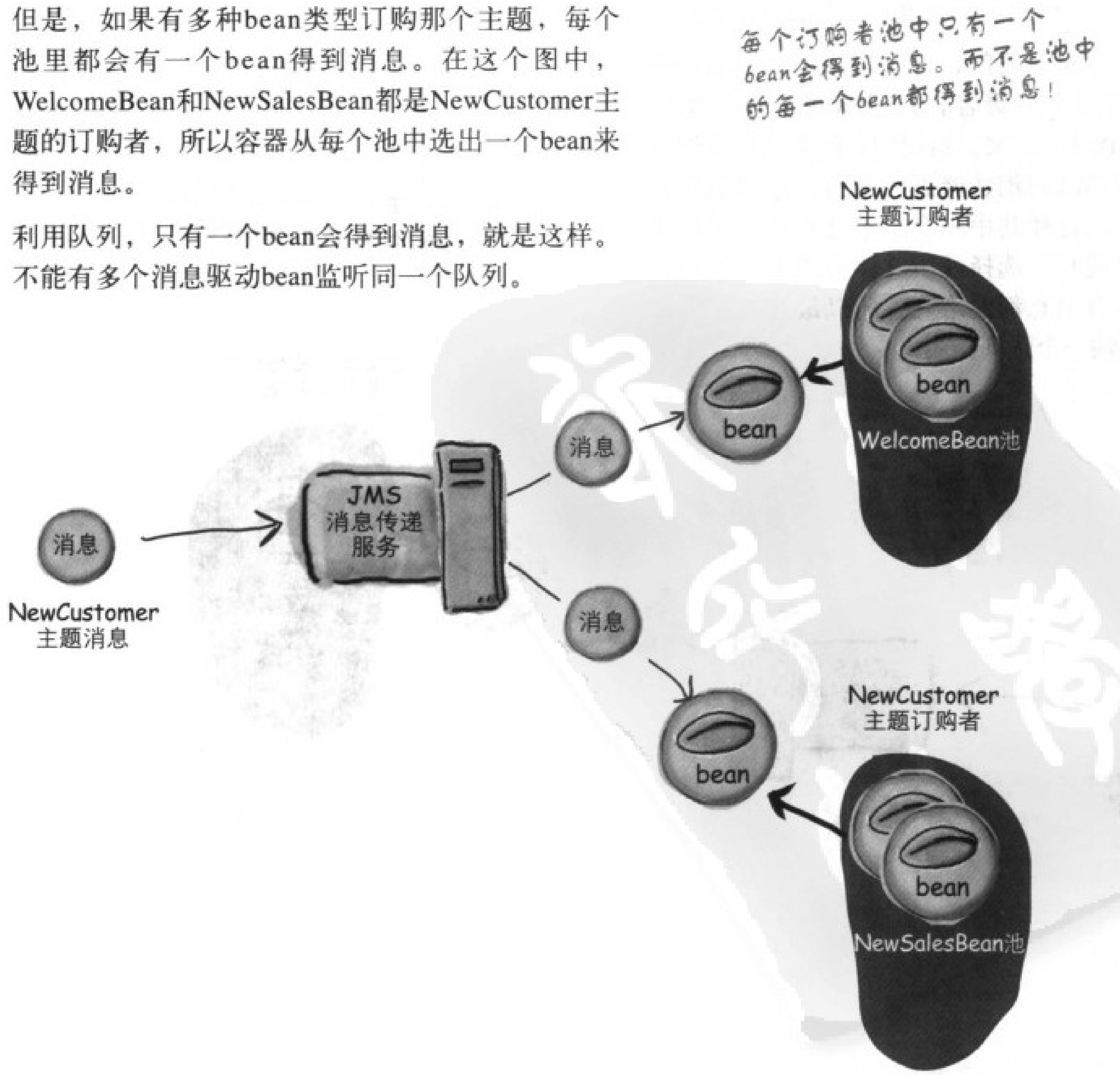
这个问题就变成，消息传递服务会把整个集群看作是应用的一个版本（而不论运行着多少个JVM），还是把每一个集群（以及它自己对应每个bean类型的复制池）看作是一个单独的监听者？

# 每个池中只有一个bean得到主题消息的一个副本

如果你有（比如说）订购NewCustomer主题的WelcomeBean，一个新消息发布给那个主题时，容器只会从WelcomeBean池中选择一个bean来得到消息。要记住，容器会为每个home维护一个单独的bean池（每个部署的bean类型都有自己的home）。容器并不是把所有消息驱动bean放到一个池里，即使它们都订购同样的主题也不例外。

但是，如果有多种bean类型订购那个主题，每个池里都会有一个bean得到消息。在这个图中，WelcomeBean和NewSalesBean都是NewCustomer主题的订购者，所以容器从每个池中选出一个bean来得到消息。

利用队列，只有一个bean会得到消息，就是这样。不能有多个消息驱动bean监听同一个队列。

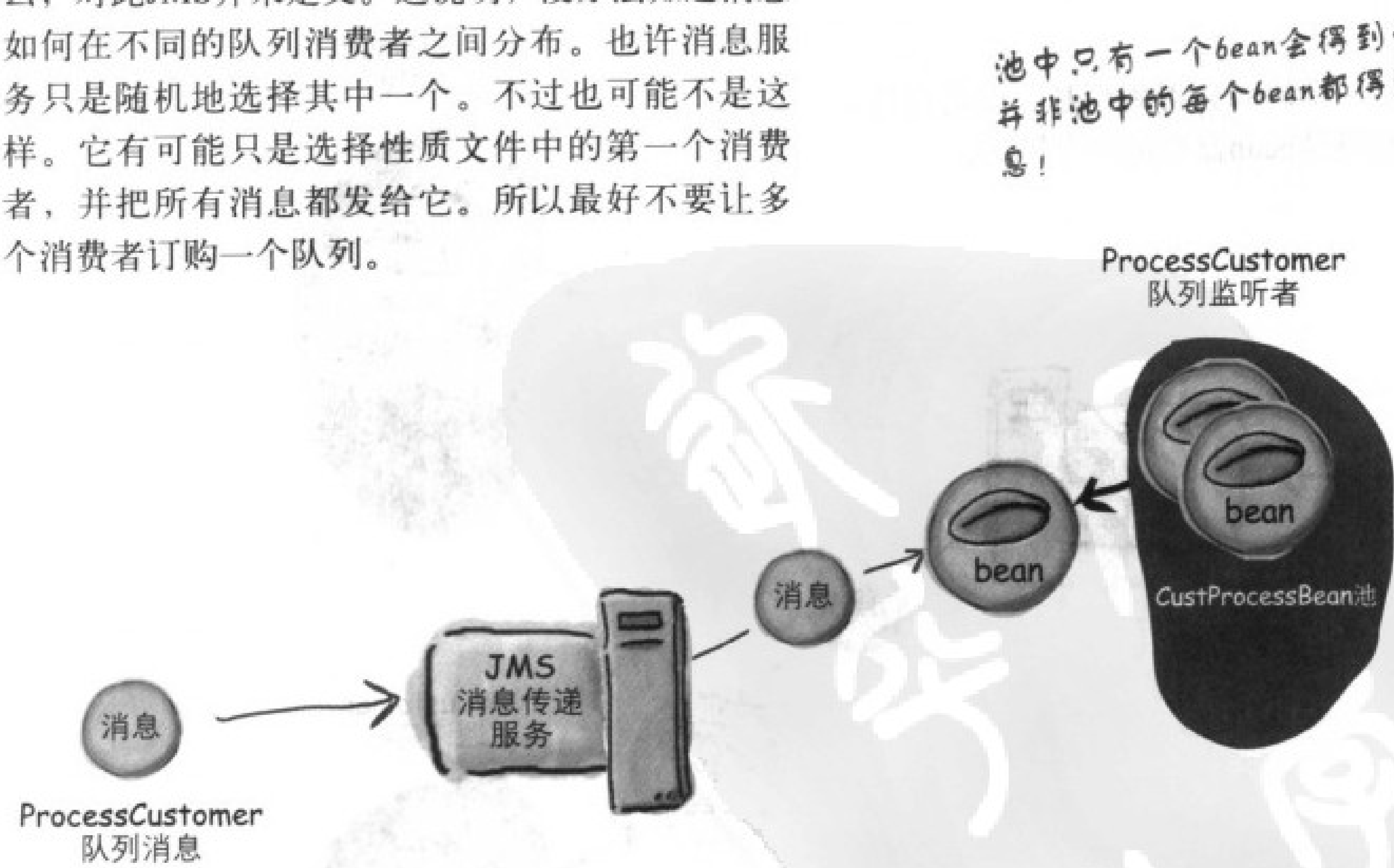


# 利用队列，只有一个bean得到消息。仅此而已。

如果有一个消息队列，其中的消息表示要处理一个新顾客，那么在与该队列相关联的池中，只有一个bean会得到这个消息。仅此而已。

哦，你不要想着问我这样一个问题：如果与这个队列关联有多个bean类型（池），会怎么样呢？

不过，如果你执意要问，可以告诉你，如果一个特定队列有多个消费者，这种情况下会发生什么，对此JMS并未定义。这说明，没办法知道消息如何在不同的队列消费者之间分布。也许消息服务只是随机地选择其中一个。不过也可能不是这样。它有可能只是选择性质文件中的第一个消费者，并把所有消息都发给它。所以最好不要让多个消费者订购一个队列。



## MessageDrivenContext

服务器建立消息驱动bean与建立无状态会话bean的做法几乎完全一样。

（1）调用bean的构造函数（必须是无参数的）。

（2）调用bean的上下文设置方法。

（3）调用bean的ejbCreate()方法。

实际上，前两步对于所有bean类型都是一样的。上下文设置方法总是在bean的构造函数之后调用，而且是在ejbCreate()之前的某个时刻。（当然，如果没有客户想在数据库中插入新的实体，实体bean可能根本不会得到ejbCreate()调用。）

那么，消息驱动bean可以对它的上下文做什么呢？我们认为，应该由你来得出结论。我们敢打保票，考试中肯定有这方面的问题，也就是消息驱动bean对它的上下文可以做什么，不能做什么。实际上，你会发现这些问题在许多考试要求中都有提到，而不只是消息驱动bean的考试要求（10.1～10.4）中才涉及。有关事务、异常和安全要求的问题都涉及到消息驱动bean及其上下文。

也就是说……一定要做下面的练习！

你可能不想记住这些内容，但是如果你花点功夫好好考虑一下，完全可以得出答案。

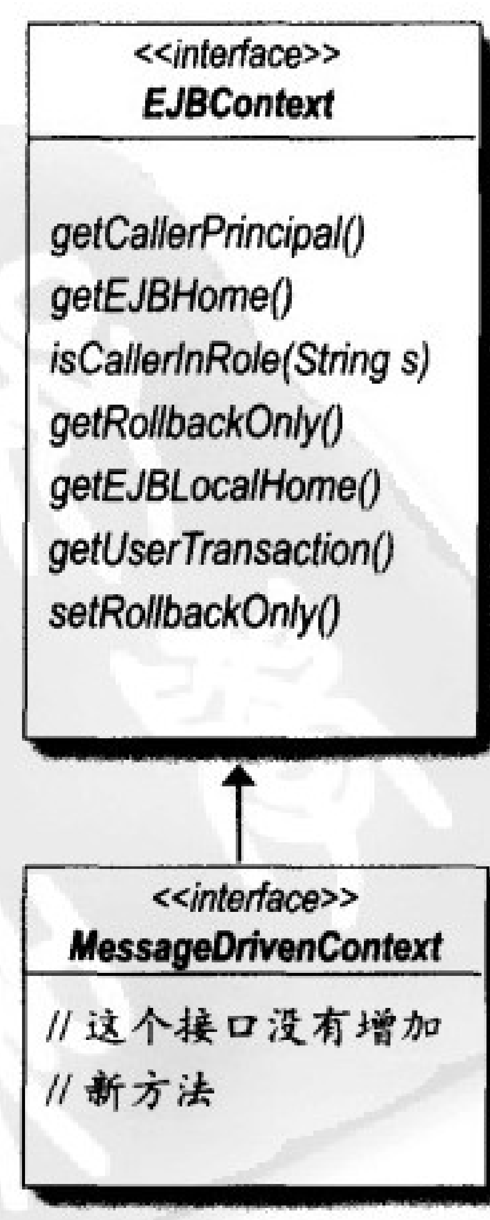
不过，下一页还是给出了答案，所以没做完之前先别翻到下一页。

### Sharpen your pencil

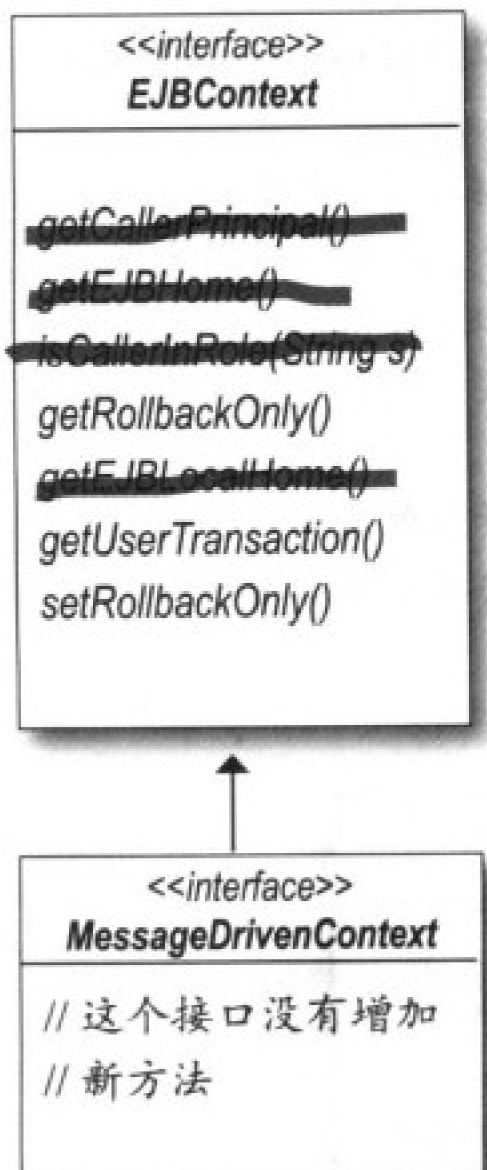
考虑一下消息驱动bean可以在它的上下文上调用哪些方法。

从EJBContext的方法列表，划掉对消息驱动bean没意义的方法（这里“没意义”的含义是指，“会导致严重的运行时异常，破坏性很大，还可能导致你作为bean开发人员被解雇，结束职业生涯”）。

哦，对了，要想好怎么解释你的答案。就当是我们坐在你后面，很严肃地要求你解释为什么会做这样的选择。



# MessageDrivenContext




setCallerPrincipal()和isCallerInRole() 给你关于客户的安全信息。但你没有客户。




这很简单。消息驱动bean没有客户。这意味着它们没有客户视图，所以没有home接口。

而且因为没有客户，也就没有客户安全信息。

因此，不能调用获得home的两个方法（因为没有home），也不能调用获得调用者安全信息的两个方法。

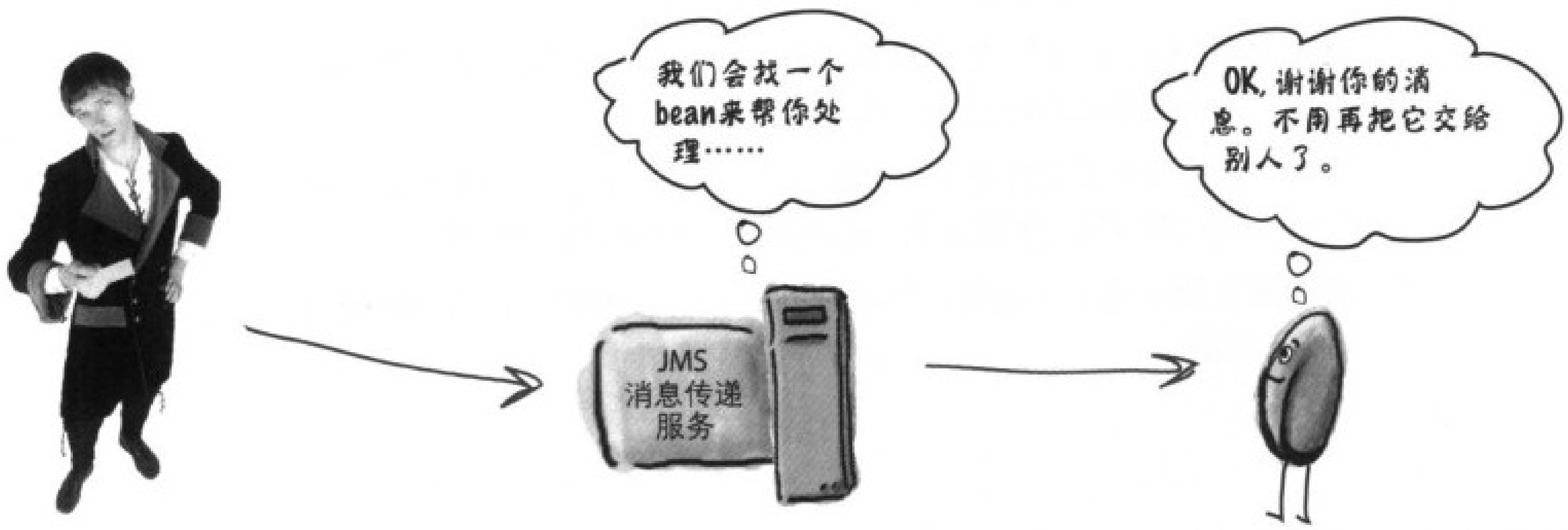
## there are no Dumb Questions

**问：** 你一直在说消息驱动bean没有客户。但是确实有人调用bean的方法，对不对？它们总不会自己调用自己吧……

**答：** 从技术上讲，确实如此，bean的方法是得到了调用。但是我们不认为容器是一个客户。容器应该是老板，而不是顾客。所以，从Java的角度讲，容器确实是bean的调用者，但它不是一个客户。至于得到安全信息，如果bean连它自己的容器都不相信，那问题就大了。这个bean肯定是个偏执狂。

# 如果出了问题会怎么样？

一切正常时……




突然……



# 消息确认

你不希望消息丢失。如果队列上有一个很重要的消息，而得到这个消息的bean无法提交，或者更糟糕的，这个bean抛出一个异常，然后死掉了，那你的重要消息会怎么样呢？

这就是确认的意义所在。容器告诉消息服务（不是原来的生产者），这个消息已经传送，一切正常。

不过，如果后来在消费者（消息驱动bean）正在处理这个消息的时候发生了不好的事情，容器会告诉消息服务把消息再放回到队列中。

容器怎么知道出了问题呢？有两种办法，这要由你（作为bean提供者）来选择。

**事务状态**

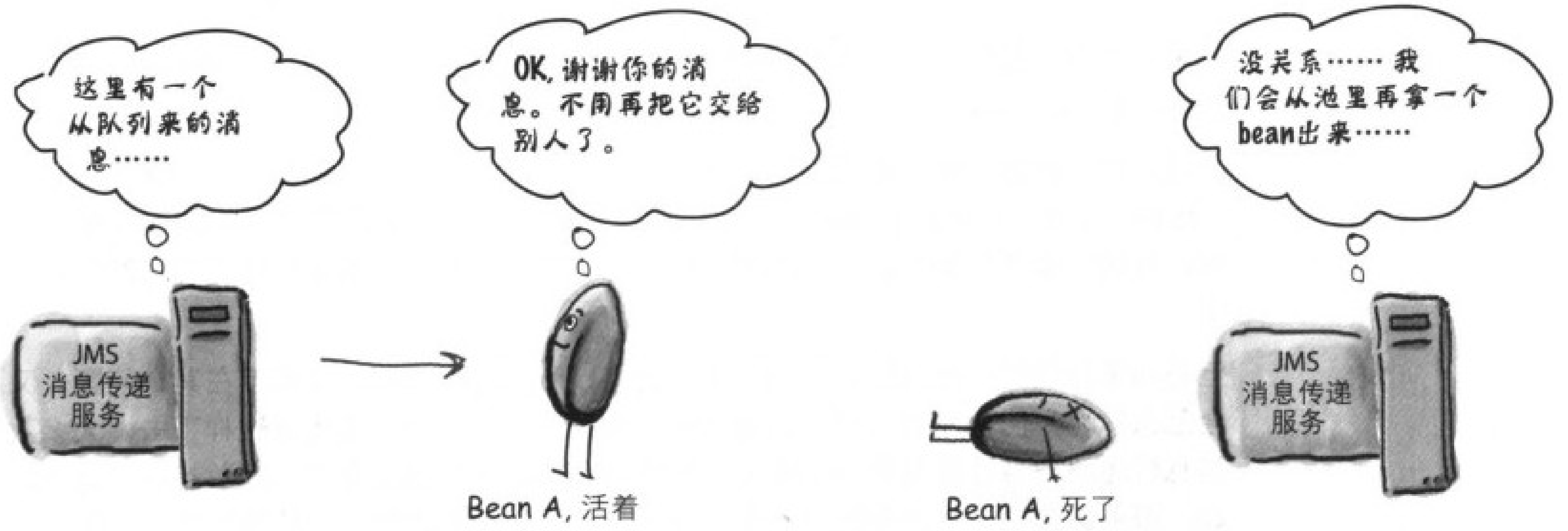
消息确认与事务提交或回滚相绑定。如果事务回滚，消息会回到队列中。使用容器托管事务划分的bean会有这种行为（我们将在下一章介绍事务）。

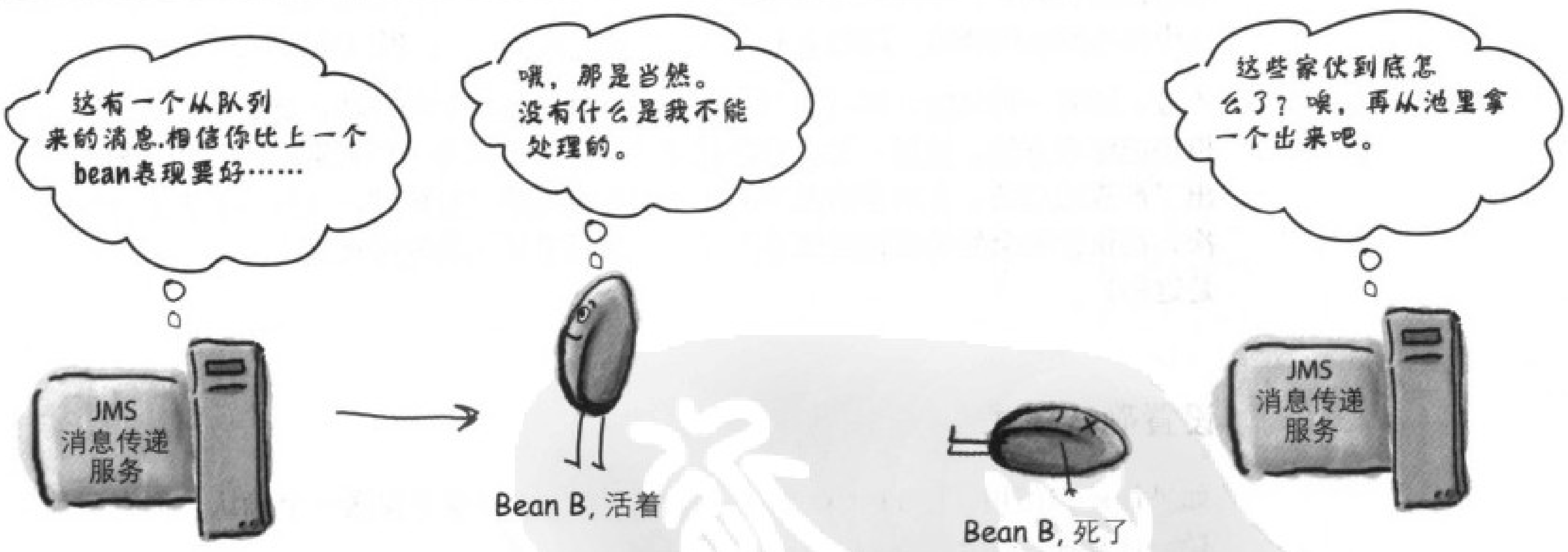
**或**

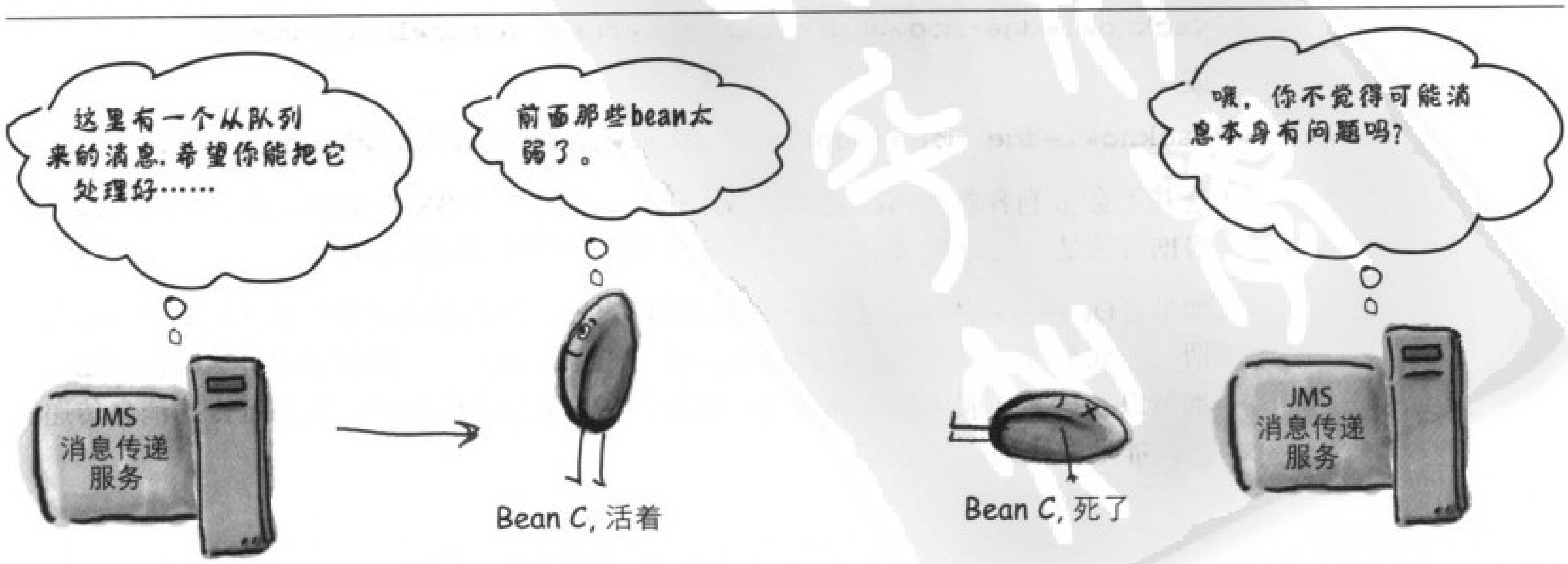
**方法完成**

消息确认与方法是否成功完成相绑定。如果方法抛出一个运行时异常，这个消息就回到队列中。使用bean托管事务的bean会有这种行为。

# 都还不错，不过先回去看看前面的情况如何结束……



# 请思考

如果你希望消息确认与事务的状态绑定，就要使用容器托管事务（container-managed transactions，CMT）。大多数情况下，这就是你的选择。

不过，如果你想把消息确认与事务状态脱离关系，惟一的选择就是使用bean托管事务（bean-managed transaction，BMT）划分。利用BMT，容器只会看你的方法是否结束。方法结束了？那好，可以做出消息确认。抛出运行时异常了？那消息就回到队列中。

但是如果有这样一种情况，bean永远不提交，而这是因为消息内部存在某个问题，又会怎么样呢？使用CMT的话，消息会不停地到来……但并不是永远这样下去。大多数消息传递服务会让你配置一个次数，如果一个特定消息尝试提交的次数超过了这个限数，服务就会说，“这个消息看上去像是个毒药；把它送到一个特殊的地方去吧（比如坏消息队列），在这里它就不会祸害别人了”。这一点你确实需要当心，因为从规范中得不到任何帮助。这完全由你的服务器开发商决定，所以你要问开发商！

不过，还有一种可能。利用BMT，你可以这样来编写你的方法，就是即使方法回滚，但仍能结束方法。这样一来，容器就会“愉快”地认为一切正常。它完全不知道已经出了严重的问题。在这种情况下，你要捕获出现的所有异常，自行回滚事务（再说一次，很快你就会知道如何回滚事务了），然后看似成功地结束这个方法（至少对容器是这样）。

## 设置确认模式

如果你确实使用了BMT，对于容器如何向消息传递服务发送一个确认，你有两种选择。这两种选择分别是：

**<acknowledge-mode>Auto-acknowledge</acknowledge-mode>**

或

**<acknowledge-mode>Dups-ok-acknowledge</acknowledge-mode>**

这并不会影响容器如何决定是否做出确认，对于一个BMT bean，是否做出确认总是根据方法是否完成来决定。确认模式只是改变容器如何发送确认。

如果是Dups-ok-acknowledge，就是在告诉容器，向消息服务做出确认时有点慢也无所谓……如果得到重复的消息你也不在乎（“dups-ok”）。这样容器什么时候做出确认都可以，这会使服务器更为高效，但是存在重复消息的危险，因为可怜的消息传递服务未能足够快地得到确认。




1. 填空，必须在bean中插入哪些代码才能成为一个合法的消息驱动bean类。

```
package headfirst;

import javax.ejb.*;
import javax.jms.*;

public class FooListenerBean ________________________________________ {

   private MessageDrivenContext context;



   public void ______________________

   public void ______________________

   public void setMessageDrivenContext(MessageDrivenContext ctx) {
       context = ctx;
   }

   public void ________________________
}
```

2. 有两个方法，状态会话bean可以在它的SessionContext上调用，但消息驱动bean不能在它的MessageDrivenContext上调用这两个方法，请把它们列出来。

____________________

____________________



# 答案

1. 填空，必须在bean中插入哪些代码才能成为一个合法的消息驱动bean类。

```
package headfirst;

import javax.ejb.*;
import javax.jms.*;

public class FooListenerBean  implements MessageDrivenBean, MessageListener  {

   private MessageDrivenContext context;



   public void  ejbCreate() { }

   public void  ejbRemove() { }


   public void setMessageDrivenContext(MessageDrivenContext ctx) {
       context = ctx;
   }

   public void  onMessage(Message msg) { }

}
```

2. 有两个方法，有状态会话bean可以在它的SessionContext上调用，但消息驱动bean不能在它的MessageDrivenContext上调用这两个方法，请把它们列出来。

调用isCallerInRole()（没有客户！）

调用getEJBHome()（没有客户视图，所以没有home接口或组件接口）

# COFFEE CRAM 模拟测验

**1** 关于消息驱动bean，下面哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 消息驱动bean有一个home接口，但没有组件接口
- ❑ B. 客户不知道消息驱动bean的身分
- ❑ C. 客户把消息驱动bean看作是一个JavaMail消息消费者
- ❑ D. 消息驱动bean的生命期由容器控制

**2** 消息驱动bean类中或其某个超类中必须实现哪些接口？（选出所有正确的答案。）

- ❑ A. **javax.jms.Message**
- ❑ B. **javax.jms.MessageListener**
- ❑ C. **javax.jms.MessageConsumer**
- ❑ D. **javax.ejb.MessageDrivenBean**
- ❑ E. **javax.ejb.MessageDrivenContext**

**3** 以下哪个方法列表按正确的顺序列出了消息驱动bean生命周期中调用的方法？(选出所有正确的答案。)

- ❑ A. **ejbCreate(), newInstance(),setMessageDrivenContext(), onMessage()**
- ❑ B. **onMessage(), newInstance(), ejbCreate(), setMessageDrivenContext()**
- ❑ C. **newInstance(), setMessageDrivenContext(), ejbCreate(), onMessage()**
- ❑ D. **newInstance(), ejbCreate(), setMessageDrivenContext(), onMessage()**
- ❑ E. **ejbCreate(), setMessageDrivenContext(),newInstance(), onMessage()**

**4** 哪些是消息驱动bean中方法的合法签名？（选出所有正确的答案。）

❑ A. `public void onMessage()`

❑ B. `public void ejbCreate()`

❑ C. `public static void onMessage()`

❑ D. `public void ejbCreate(javax.jms.Message m)`

❑ E. `public void onMessage(javax.jms.Message m)`

❑ F. `public void onMessage(javax.jms.Message m) throws java.rmi.RemoteException`

**5** 消息驱动bean什么时候能够通过JNDI访问java:comp/env?

❑ A. `ejbCreate()`

❑ B. `ejbRemove()`

❑ C. `setMessageDrivenContext()`

❑ D. 以上都不对

**6** 哪些消息驱动bean方法取一个参数？（选出所有正确的答案。）

❑ A. `ejbCreate()`

❑ B. `ejbRemove()`

❑ C. `onMessage()`

❑ D. `setMessageDrivenContext()`

**7** 消息驱动bean什么时候能够访问其他企业bean?

❑ A. `ejbCreate()`

❑ B. `ejbRemove()`

❑ C. `onMessage()`

❑ D. `setMessageDrivenContext()`

❑ E. 以上都不对

**8** 关于容器对消息驱动bean的支持，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 容器必须支持将消息驱动bean部署为一个JavaMail队列的消费者
- ❑ B. 容器不必支持消息驱动bean的事务作用域
- ❑ C. 容器可以保证先进的消息先处理
- ❑ D. 容器必须确保bean实例是不可再入的

**9** 使用BMT划分的消息驱动bean什么时候能够访问资源管理器？

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ❑ C. `onMessage()`
- ❑ D. `setMessageDrivenContext()`
- ❑ E. 以上都不对

**10** 关于消息驱动bean的消息确认，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 消息确认模式不能以声明的方式定义
- ❑ B. 应当将JMS API用于消息确认
- ❑ C. 对于BMT bean，容器使用acknowledge-mode部署描述文件元素
- ❑ D. 对于CMT bean，容器使用acknowledge-mode部署描述文件元素

**11** 关于消息驱动bean的部署描述文件，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. bean提供者必须保证bean与一个特定的队列或主题相关联
- ❑ B. 部署描述文件可以指示一个bean是面向主题还是面向队列
- ❑ C. 可以指示一个Queue类型的bean是否应该支持持久订购
- ❑ D. 把多个bean与同一个JMS队列关联是合适的

# COFFEE CRAM 模拟测验答案

（规范：311～312）

**1** 关于消息驱动bean，下面哪些说法是正确的？（选出所有正确的答案。）

- ☐ A. 消息驱动bean有一个home接口，但没有组件接口— MDB没有客户视图
- ☑ B. 客户不知道消息驱动bean的身分
- ☐ C. 客户把消息驱动bean看作是一个JavaMail消息消费者
- ☑ D. 消息驱动bean的生命期由容器控制

（规范：314～315）

**2** 消息驱动bean类中或其某个超类中必须实现哪些接口？（选出所有正确的答案。）

- ☐ A. **`javax.jms.Message`**
- ☑ B. **`javax.jms.MessageListener`** — onMessage()方法在这个接口中定义
- ☐ C. **`javax.jms.MessageConsumer`**
- ☑ D. **`javax.ejb.MessageDrivenBean`**
- ☐ E. **`javax.ejb.MessageDrivenContext`**

（规范：319）

**3** 以下哪个方法列表按正确的顺序列出了消息驱动bean生命周期中调用的方法？（选出所有正确的答案。）

- ☐ A. **`ejbCreate(), newInstance(),setMessageDrivenContext(), onMessage()`**
- ☐ B. **`onMessage(), newInstance(), ejbCreate(), setMessageDrivenContext()`**
- ☑ C. **`newInstance(), setMessageDrivenContext(), ejbCreate(), onMessage()`**
- ☐ D. **`newInstance(), ejbCreate(), setMessageDrivenContext(), onMessage()`**
- ☐ E. **`ejbCreate(), setMessageDrivenContext(),newInstance(), onMessage()`**

**4** 哪些是消息驱动bean中方法的合法签名？（选出所有正确的答案。） （规范：324）

- ❑ A. `public void onMessage()`
- ☑ B. `public void ejbCreate()`
- ❑ C. `public static void onMessage()`
- ❑ D. `public void ejbCreate(javax.jms.Message m)`
- ☑ E. `public void onMessage(javax.jms.Message m)`
- ❑ F. `public void onMessage(javax.jms.Message m) throws java.rmi.RemoteException` —绝对不能声明RemoteException

**5** 消息驱动bean什么时候能够通过JNDI访问java:comp/env? （规范：320）

- ☑ A. `ejbCreate()`
- ☑ B. `ejbRemove()`
- ☑ C. `setMessageDrivenContext()`
- ❑ D. 以上都不对

**6** 哪些消息驱动bean方法取一个参数？（选出所有正确的答案。） （规范：319，324）

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ☑ C. `onMessage()` — 取一个消息
- ☑ D. `setMessageDrivenContext()` — 取一个上下文

**7** 消息驱动bean什么时候能够访问其他企业bean? （规范：320）

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ☑ C. `onMessage()` — 只有在带“一个有意义的事务上下文”的方法中才能访问
- ❑ D. `setMessageDrivenContext()`
- ❑ E. 以上都不对

**8** 关于容器对消息驱动bean的支持，以下哪些说法是正确的？（选出所有正确的答案。） （规范：325～326）

❑ A. 容器必须支持将消息驱动bean部署为一个JavaMail队列的消费者

❑ B. 容器不必支持消息驱动bean的事务作用域

❑ C. 容器可以保证先进的消息先处理

☑ D. 容器必须确保bean实例是不可再入的

**9** 使用BMT划分的消息驱动bean什么时候能够访问资源管理器？ （规范：320）

❑ A. **ejbCreate()**

❑ B. **ejbRemove()**

☑ C. **onMessage()** —要有一个"有意义的事务上下文"

❑ D. **setMessageDrivenContext()**

❑ E. 以上都不对

**10** 关于消息驱动bean的消息确认，以下哪些说法是正确的？（选出所有正确的答案。） （规范：317）

❑ A. 消息确认模式不能以声明的方式定义

❑ B. 应当将JMS API用于消息确认

☑ C. 对于BMT bean，容器使用acknowledge-mode部署描述文件元素 —可以是Auto—acknowledge或Dups—ok—acknowledge

❑ D. 对于CMT bean，容器使用acknowledge-mode部署描述文件元素

**11** 关于消息驱动bean的部署描述文件，以下哪些说法是正确的？（选出所有正确的答案。） （规范：317）

❑ A. bean提供者必须保证bean与一个特定的队列或主题相关联 —这是部署人员的任务

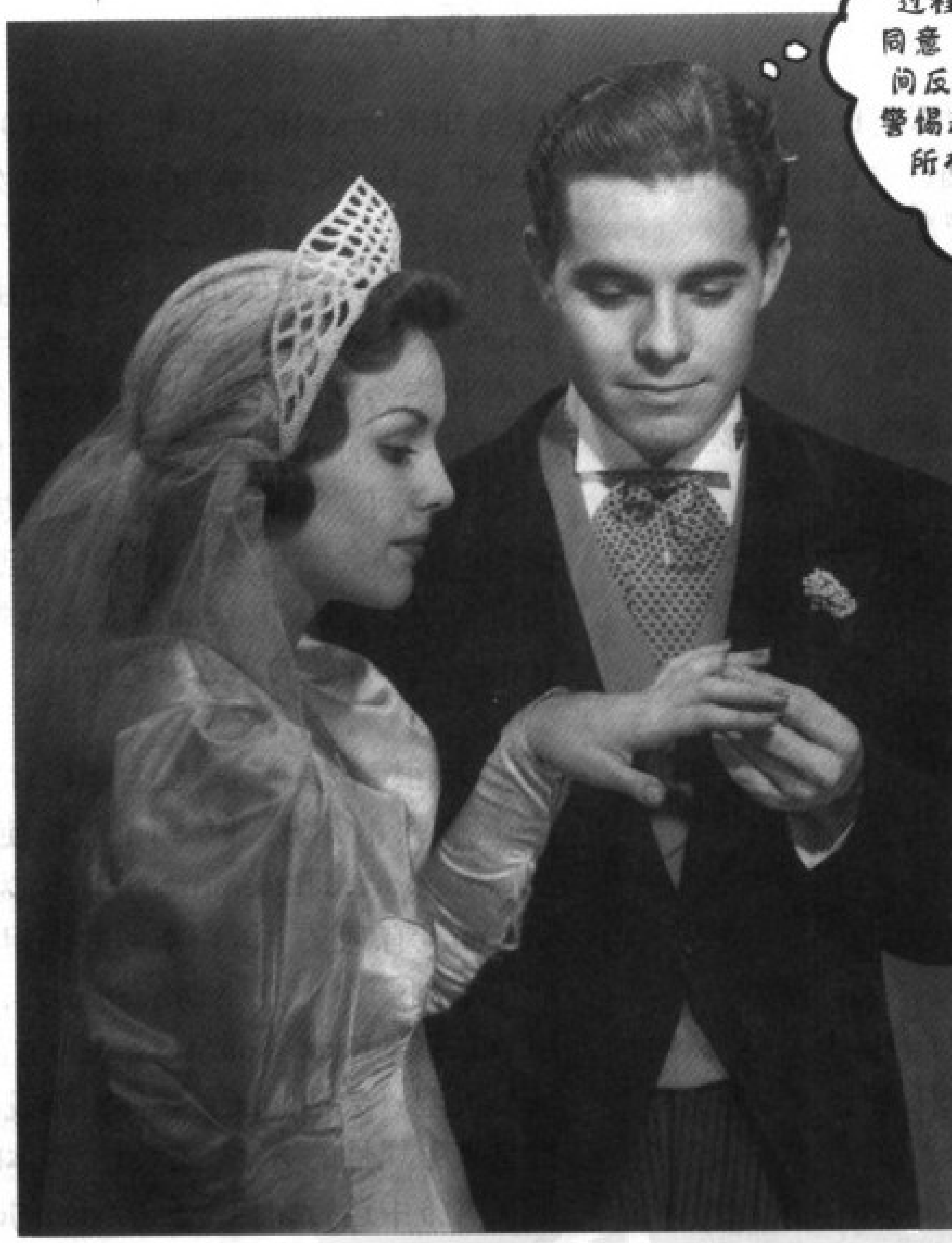
☑ B. 部署描述文件可以指示一个bean是面向主题还是面向队列

❑ C. 可以指示一个Queue类型的bean是否应该支持持久订购 —持久订购只用于主题！

❑ D. 把多个bean与同一个JMS队列关联是合适的

# 9 EJB事务

## 原子时代




**事务可以保护你。**利用事务，可以冒点险。你可以尝试做些大事，因为你知道即使过程中出了意外，也可以假装什么事都没有发生。一切又能恢复到从前。想法很简单，要么提交事务中的所有一切，要么回滚，这样谁也不会看到你原来打算做（但失败）的事情。EJB中的事务是一大妙笔，你可以部署一个有定制事务行为的bean，而不用去动这个bean的源代码！但是如果需要，也确实可以编写事务代码，所以这方面我们也会进行介绍。

# EJB事务

## 标准大纲：

11.1 有关EJB事务，区别哪些说法或示例是正确的，哪些不正确，这包括bean托管事务划分和容器托管事务划分。

11.2 有关应用组装人员的职责，区别哪些说法或示例是正确的，哪些不正确，包括如何使用与事务相关的部署描述文件元素，以及明确特定bean类型有哪些必须指定事务属性的方法。

11.3 给定一组事务行为，与适当的事务属性匹配。

11.4 给定一组职责，明确哪些是容器关于事务（包括容器托管事务划分和bean托管事务划分）的职责，包括getRollbackOnly、setRollbackOnly、getUserTransaction、SessionSynchronization回调的处理。

（注意：11.4中有一部分涉及有关会话bean的一章中的SessionSynchronization）。

## 具体含义：

需要知道bean托管事务（BMT）划分和容器托管事务（CMT）划分的规则和影响。例如，必须知道消息驱动bean和会话bean既可以使用CMT也可以使用BMT，但是实体bean只能使用CMT。而且使用BMT的消息驱动bean和无状态会话bean必须在方法完成之前结束事务，但是有状态会话bean可以保持事务经过客户的多个方法调用期间一直打开。你要知道BMT中的事务传播是单向的：BMT事务可以随bean的方法调用传播出去（也就是说，被调用的方法会使用这个事务），但是现有的事务上下文不能传播到一个BMT bean中去。换句话说，BMT bean只能在bean本身开始的事务中运行。

你必须很清楚事务属性对CMT的影响。例如，应该了解到消息驱动bean只能使用6个属性中的两个（NotSupported和Required），因为其他4个属性对消息驱动bean没有意义（调用事务即调用方法时所处的事务（不能传播到一个消息驱动bean中去，因为只有容器才会调用消息驱动bean的onMessage()方法）。还必须知道每种bean类型（会话bean、实体bean或消息驱动bean）中必须指定事务属性的方法。例如，实体bean的create()方法就必须在事务中运行，但是会话bean的create()方法则在“未指定的事务上下文”中运行。你必须会在部署描述文件中指定事务属性。

最后一点，你要知道bean在事务中时能从容器得到些什么。例如，必须知道，如果在bean的上下文上调用getRollbackOnly，那么无论在什么情况下容器都不能提交事务。

## EJB事务是一个原子工作单元。

事务意味着你把一些工作（语句、方法调用、整个方法、访问数据库等等）以某种方式包装到一个单元中，这样要么所有一切都成功，要么一切都恢复到其原来的状态。

换句话说，整个原子单元要么提交，要么回滚。

它要么正常工作，要么我们完全忘掉刚才发生的一切。



## 购物车结账：<br>这是一个经典的EJB事务例子。

假设你有一个在线购物车系统。该结账了，需要做些什么呢？至少是：

- 用户要**确认**订单
- 验证用户的**信用卡**，并完成支付
- 从**库存**中删除已经购买的商品
- 创建和提交一个发货**订单**

这些都要作为一个单元发生，如果某个工作不成功，那么所有工作都应该回滚，就好像它们从没有发生过一样……

如果信用卡非法，就不希望从库存中删除商品。如果商品没有库存，就不希望提交一个发货订单。如果用户没有确认订单，那么所有这些都不该发生！如果其中某个工作出了问题，你希望事务能回滚，而不是提交。如果不能做事务回滚，想想看会经受多大的恐慌。假设你已经走完了4步中的前3步，最后却发现用户根本没有确认订单。你就必须退回去，把钱再加回到用户的信用卡上，撤销订单，把商品放回到库存中。

Relax

**你不用了解JTS、XA或除javax.transaction.UserTransaction以外的其他事务API。**

考试不会考任何底层事务API细节。例如，你不用知道服务器/容器如何与一个事务资源（如数据库）通信。另外，尽管EJB支持分布式事务，但你不会遇到这样的问题，考试时不会问你这是如何工作的。我们也知道，这就显不出你已经掌握了两段提交协议，对此你可能会有点失望。



# ACID测试

## 你的事务安全吗？

100%的专家（以及行业界的所有人）都一致同意，一个好的、安全的事务要有4个特性（不仅在EJB中如此，在Gosling还没有发明Java之前，ACID测试就早已经存在了）。你的事务要想通过ACID测试，必须保证以下几点：

### 原子性（Atomic）

要么所有都成功，要么所有都失败（并回滚）。

如果其中有些工作能提交，而另外一些没有提交，这个事务就不是原子的。

### 一致性（Consistent）

无论是成功（提交）还是失败（回滚），数据都应该与业务逻辑实际保持一致。比如说，如果你能把商品从库存中取出来，但是没有真正提交订单，就会遇到麻烦。最后就会出现这种情况：尽管实际库存中（也就是说，在某个地方的仓库里）有商品，但是从计算机的显示上却看不出来（因为系统已经删除了这些记录）。

### 独立性（Isolated）

假设你运行了两个不同的事务，可能会访问同一个数据库。你不希望一个事务的影响会破坏另一个事务的状态。换句话说，事务相互之间应该有所保护（独立）。独立性与线程同步很类似，你不希望一个事务读数据（想对它做操作）的时候该数据正处在另一个事务的处理当中（而且这个事务尚未提交它对数据所做的修改）。

### 持久性（Durable）

一旦事务提交，该事务所做的修改就必须是永久的！就算是服务器瘫痪了，也必须恢复回来，完成它原来的提交。

## 分布式事务：两段提交

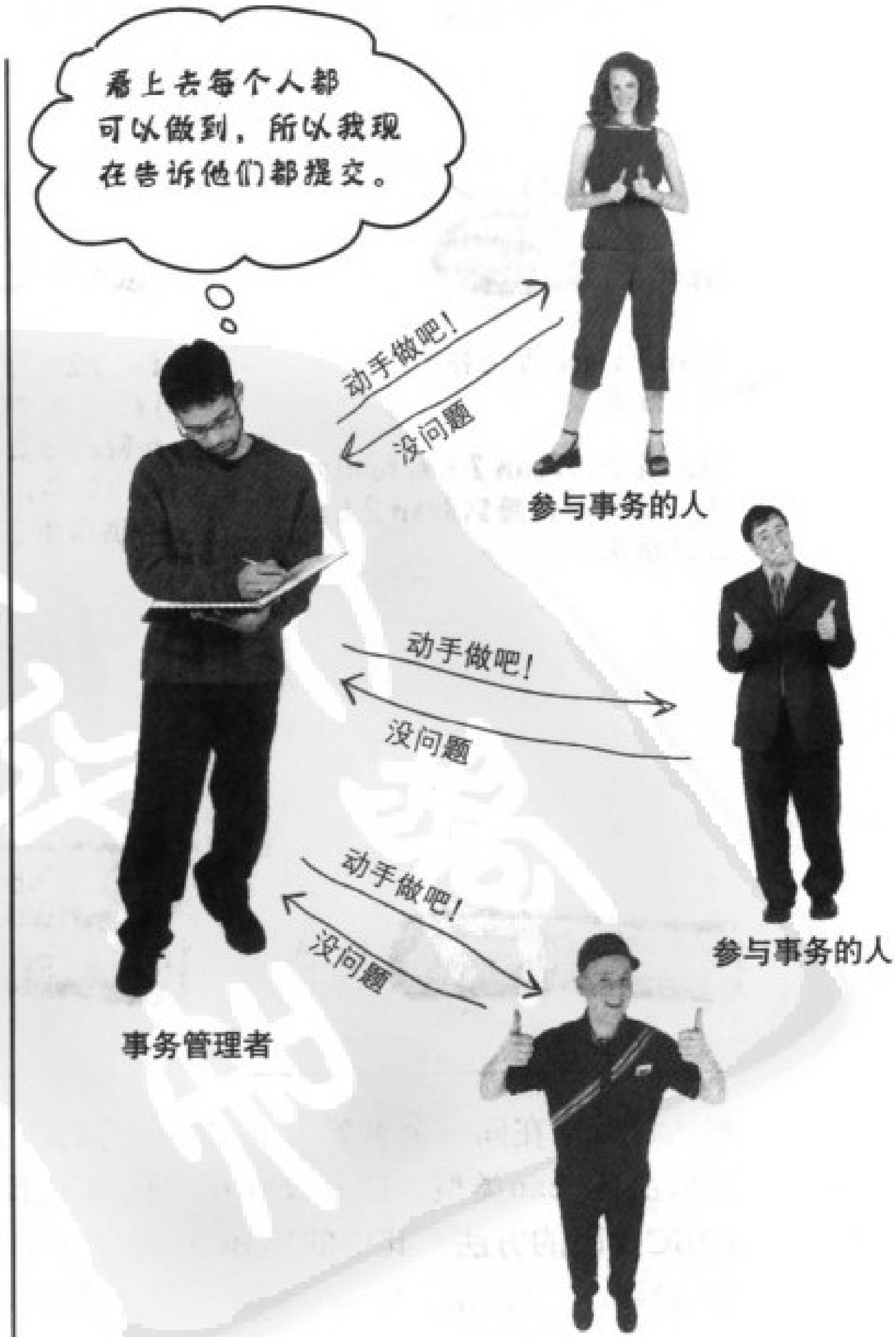



大多数EJB容器都通过一种两段提交协议来支持分布式事务。如果你是一个事务管理者（如J2EE服务器），你可能管理着很多人，包括一个数据库、另一个bean，还有网络上的另一个服务器。一旦你告诉每一个人都提交，要想撤销就不那么容易了，所以在给出信号要求提交之前，需要先确保每一个人都能完成你要求的事情。作为事务管理者，你的任务就是看是不是每个人都准备好完成工作（更新数据库、从账户取钱，等等），然后，根据这些结果，告诉他们去做各自的工作（提交），或者告诉他们忘掉这个任务（回滚）。

第1阶段




第2阶段



# EJB中是如何做到的

## 事务可以通过方法调用传播

一个bean在一个事务中运行代码，而且调用了另一个bean上的一个方法，此时有3种可能的情况：

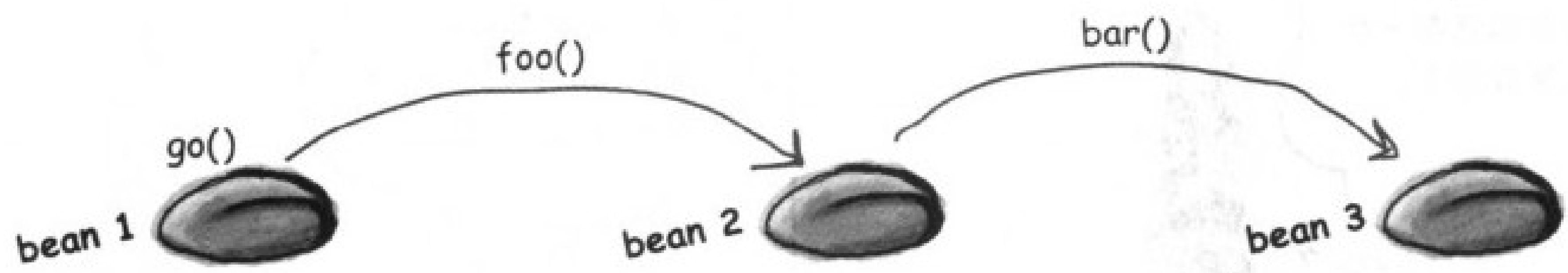
A）被调用的方法在调用者所在的事务中运行。

B）被调用的方法无事务运行（在无事务的条件下运行）。

C）被调用的方法在它自己的新事务中运行。

---

Ⓐ 第一个方法中开始的事务传播到调用栈中所有其他方法。所有被调用的方法都在同一个事务中运行（在本书中，我们把事务“transaction” 简写作“tx”）。



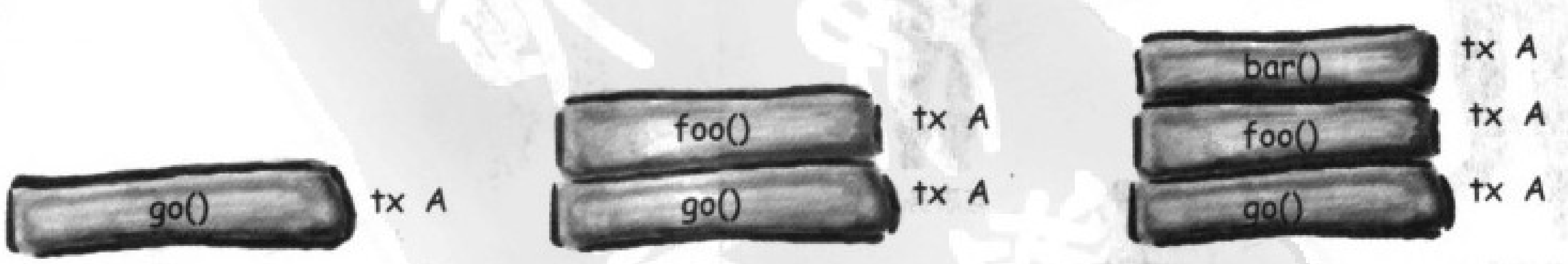

Bean 1的go()方法开始一个事务（tx A）。

Bean 1调用Bean 2上的foo()方法，事务则传播到Bean 2上的方法调用中。

Bean 2的foo()方法在来自Bean 1的事务（tx A）中运行，并调用Bean 3上的方法bar()。这个事务（tx A）传播到Bean 3上的方法调用中。

Bean 3的bar()方法在来自Bean 2的事务（tx A）中运行。

如果事务A中任何方法发生一个系统异常（如EJBException），整个事务都会回滚。




---

那么，对于在同一个事务上下文中运行的多个方法来说，这意味着什么？其意义是不同的，这取决于bean类型，以及bean在其代码里都做了些什么。例如，假设Bean A有一个包括JDBC代码的方法，其中的JDBC代码对一个数据库行做了更新。如果同一个事务中的某个方法导致一个回滚，Bean A的更新就不会提交，就算数据库再想更新也无济于事。

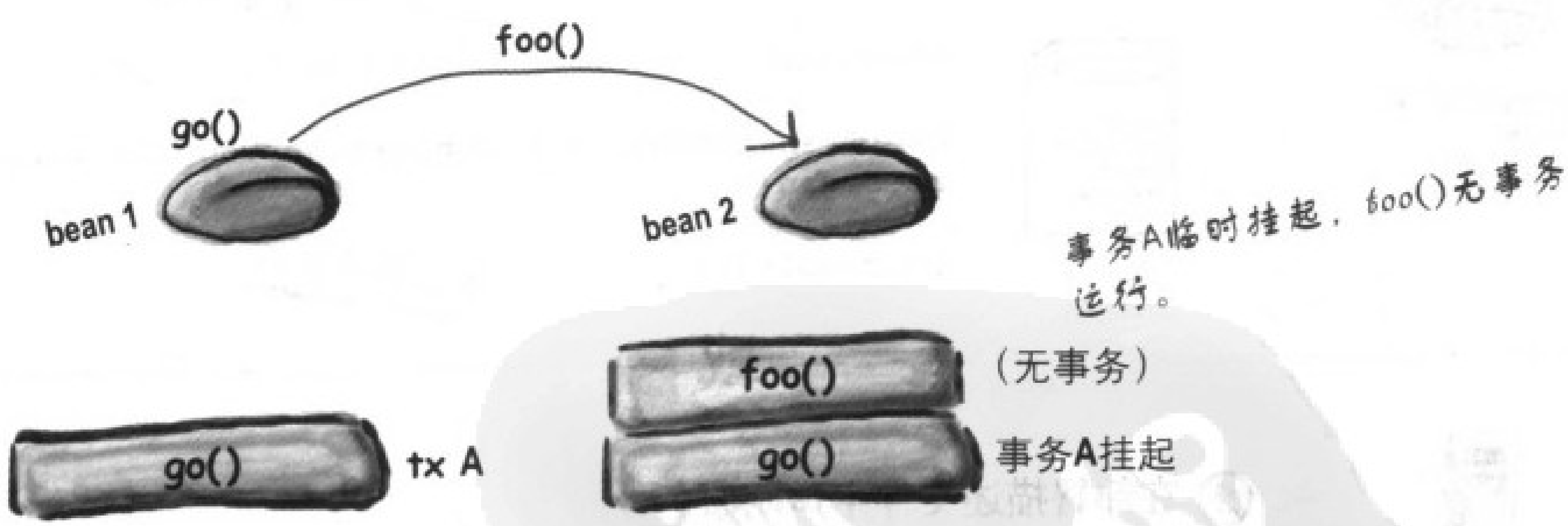
## 有些事务不传播

## 调用者的事务可以被挂起

如果一个事务性bean方法调用了另一个方法，被调用的方法（不管是同一个bean的方法，还是另外一个bean的方法）可能不在同一个事务中运行。在这种情况下，被调用的方法可能在一个全新的事务中运行，或者根本不在事务中运行（稍后我们会介绍容器如何决定是否传播事务，究竟是在无事务运行，还是开始一个新事务）。

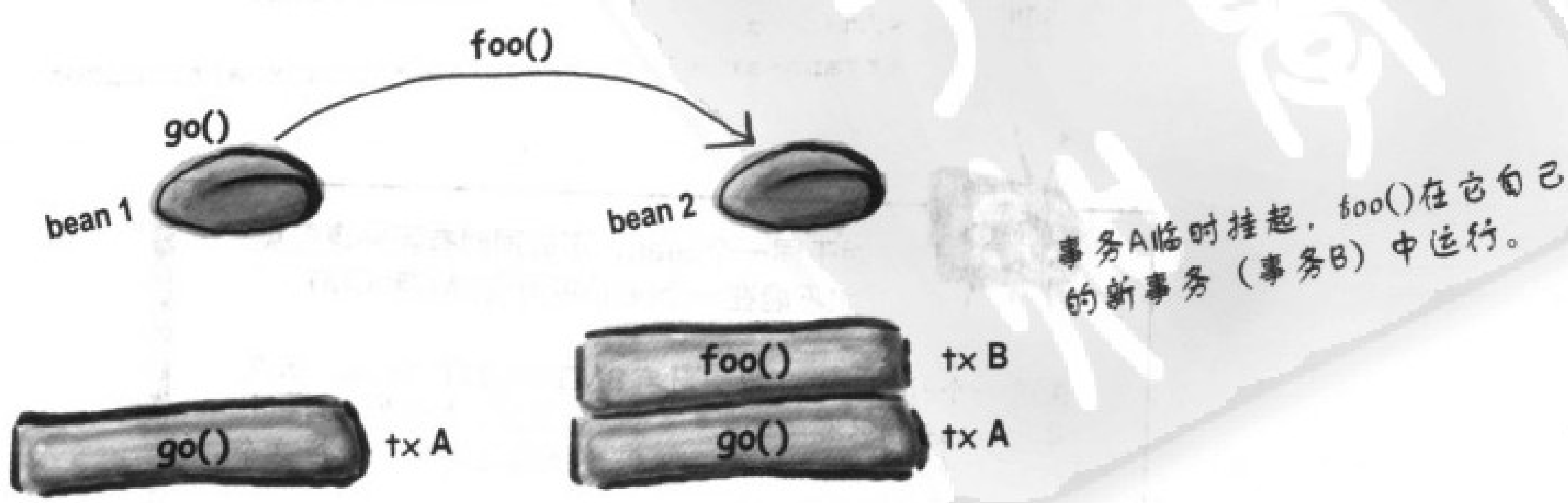
---

Ⓑ 第一个事务挂起，第二个方法在没有事务的情况下运行。




---

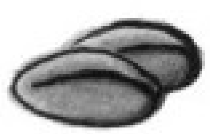
Ⓒ 第一个事务挂起，第二个方法在一个新事务中运行。



# 怎么建立（或得到）一个事务？

## 两种办法：编写事务代码，或者声明事务

容器会管理你的事务，但是你必须告诉它怎么管理。可以在bean类中放上事务代码，也可以在部署描述文件中放事务声明。到目前为止，最常用的方法是使用部署描述文件，因为这种方法更简单，还能支持bean重用，另外，对于实体bean的事务只能这样做。把事务信息放在部署描述文件中而不是代码中的话，你就可以把同一个bean部署多次，每次得到不同的事务行为，而无需修改代码！



bean托管事

(BMT)

① 在bean中编写事务代码。

告诉部署描述文件由你负责事务管理



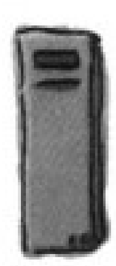

.java

```
<transaction-type>Bean</transaction-type>

UserTransaction ut = context.getUserTransaction();
ut.begin();
//  事务代码
ut.commit();
```

通过程序方式

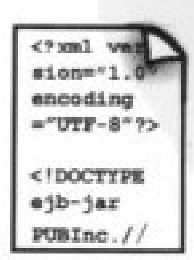
或



容器托管事

(CMT)

② 在部署描述文件中声明事务。

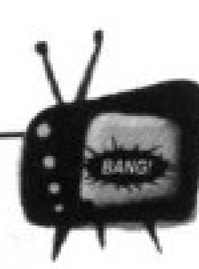



.xml

```
<transaction-type>Container</transaction-type>

<method>
   <ejb-name>MyBean</ejb-name>
   <method-name>bar</method-name>
</method>
<trans-attribute>Required</trans-attribute>
```

通过声明方式




**对于同一个bean，不能同时有两种事务类型!不能在一个bean中混合BMT和CMT。**

在同一个ejb-jar中可以同时有BMT和CMT bean，但是每个bean必须只有一种事务类型。考试时，如果你看到一个bean里有事务代码，就要确保部署描述文件中对应这个bean有这样的声明：
`<transaction-type>Bean</transaction-type>`

# 与事务相关的方法在两个接口中

**UserTransaction**

begin()
commit()
getStatus()
rollback()
setRollbackOnly()
setTransactionTimeout()

**javax.transaction.UserTransaction**

UserTransaction接口中的所有方法都是面向使用bean托管事务(BMT)的bean。

使用容器托管事务（CMT）的bean不允许调用这个接口的任何方法。

**EJBContext**

getUserTransaction()

getCallerPrincipal()
getEJBHome()
getEJBLocalHome()
isCallerInRole()

setRollbackOnly()
getRollbackOnly()

**javax.ejb.EJBContext**

EJBContext中的方法同时面向BMT和CMT bean

仅用于BMT bean的方法：
getUserTransaction()

同时适用于BMT和CMT的方法：
(所有不与事务相关的方法)

仅用于CMT bean：
getRollbackOnly()
setRollbackOnly()

（SessionContext、EntityContext和MessageDrivenContext的超类）

---

**BRAIN POWER**

请考虑CMT和BMT之间的区别。

如果非要从中选择一个而不选另一个，你会选哪一个呢？

❑ CMT　❑ BMT

为什么？它有什么优缺点？

# 建立一个BMT事务

① 从EJBContext得到一个UserTransaction引用。

```
context.getUserTransaction()
```

② 开始事务。

```
ut.begin()
```

③ 结束事务（提交或回滚）。

```
ut.commit()
ut.rollback()
```

（在后面的异常一章我们会谈到具体的异常。）

通过在EJBContext上调用getUserTransaction()，得到某个实现了UserTransaction接口的对象的引用。

```
public void checkOut() throws Exception {

    UserTransaction ut = context.getUserTransaction();


      ut.begin();


      anotherBean.validateCredit(customerNum);

      checkInventory();


      ut.commit();


      //or ut.rollback();


      doNonTxStuff();
}
```

这表示：“现在开始一个新事务。”

调用其他方法：假设这些方法都在这个新事务下运行（也可能并非如此，但我们后面再讨论这个问题）。

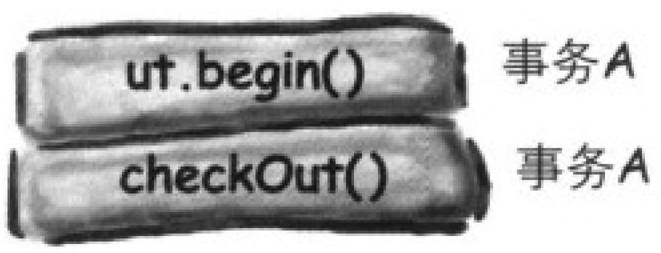
现在告诉容器，“继续干吧，把这个事务里发生的一切全部提交”，以此结束事务。

（或者，也可以通过回滚来结束事务。）

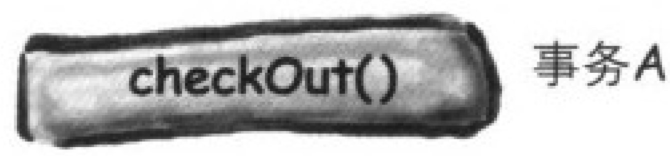
这个方法调用无需事务。从这个代码无法区别doNonTxStuff()是否开始它自己的事务。必须查看这个方法才能知道。

# checkOut()方法的调用栈

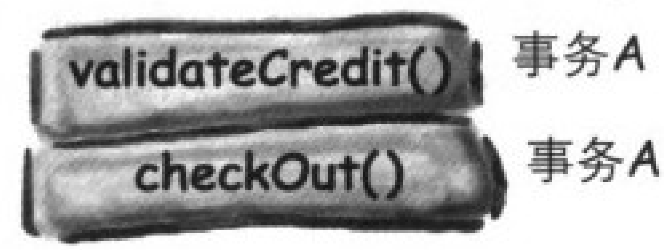
① begin()被调用，事物开始。



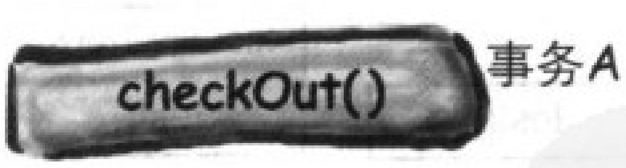

② ut.begin()结束，现在，checkout()方法在一个事物(即事物A)中运行。



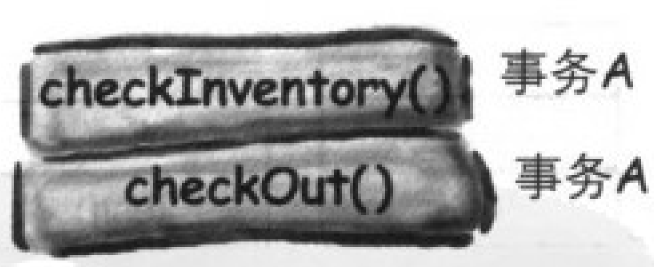

③ validateCredit()被调用，并在checkOut()所在的事物中运行。



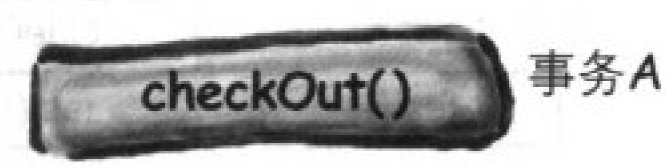

④ validateCredit()结束（从栈中弹出），checkOut()方法仍在事物A中运行。



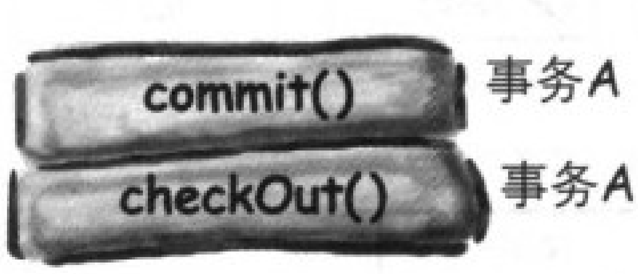

⑤ checkInventory()被调用，在同一个事物（事物A）中运行。



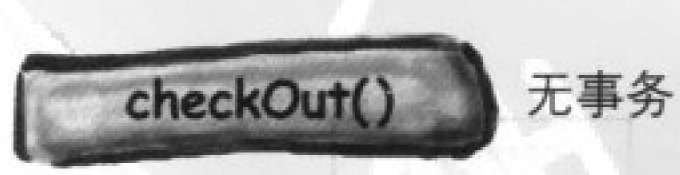

⑥ checkInventory()结束，并从栈中弹出，事物（事物A）仍打开。




⑦ commit()被调用，这会结束事物。




⑧ commit()完成，现在checkOut()在无事物的条件下运行。




⑨ doNonTxStuff()在无事物的情况下得到调用。

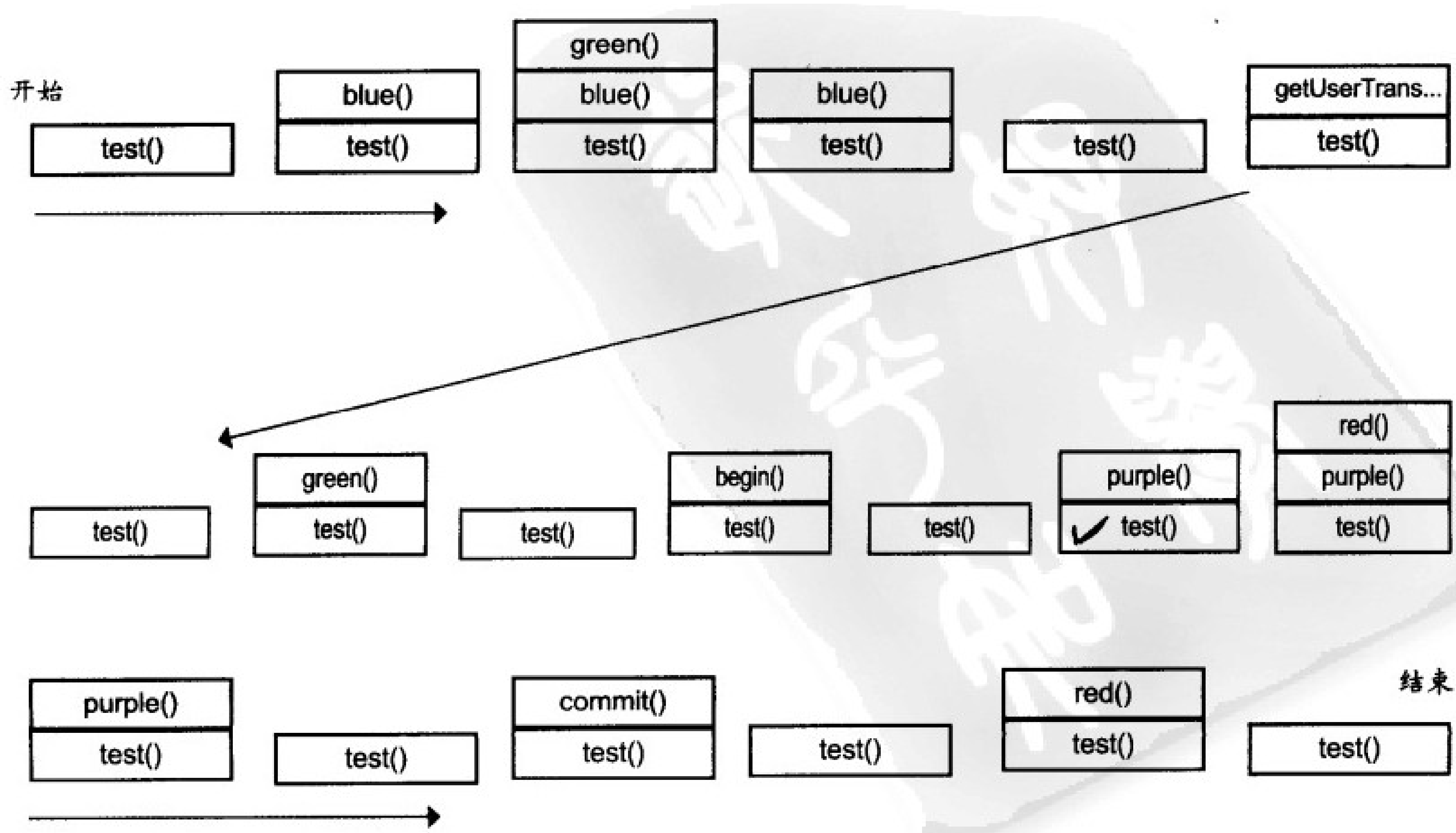


## Sharpen your pencil

利用这个代码清单，在适当的调用栈帧上打勾，也就是说，如果一个栈帧正处在一个事务中，则打勾。我们已经将其中一个符合条件的栈帧打了勾。

```
public void test() throws Exception {
    blue();
    UserTransaction ut = ctx.getUserTransaction();
    green();
    ut.begin();
    purple();
    ut.commit();
    red();
}

void blue() { green(); }
void green() { }
void purple() { red(); }
void red() { }
```



# 对BMT不能做的事情

1. **BMT bean在结束当前事务之前不能开始另一个事务。**

```
public void go() {
    UserTransaction ut = context.getUserTransaction();
    ut.begin();
    doStuff();
    ut.begin();
    // 更多
}
```

千万别这样！前一个事务没有结束（提交或回滚）的时候不能开始另一个事务。如果真的可以这样做，就表示你可以有"嵌套事务"了，而这是不行的。

**EJB中是不允许嵌套事务的！**

你要知道"嵌套事务"一词的含义，以及在代码中嵌套事务会表现为什么样子。

**BRAIN POWER**

想像一下，在结束当前事务之前开始一个新事务会有怎样的影响。如果确实允许这样做可能出现什么情况？

---

2. **BMT无状态会话bean和消息驱动bean在结束事务之前不能结束一个事务性方法。**

```
public void go() {
    UserTransaction ut = context.getUserTransaction();
    ut.begin();
    doMore();
}
```

这里存在问题！我们结束了方法，但是没有结束前面开始的事务。换句话说，这里没有commit()或rollback()。

**只是有状态会话bean可以在事务打开的情况下结束方法。**

**BRAIN POWER**

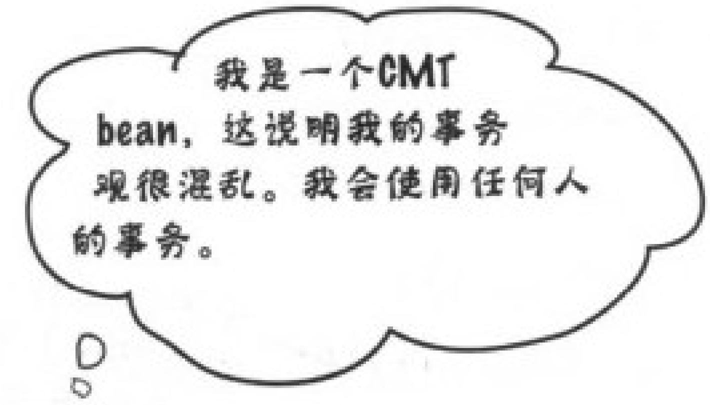
为什么有状态会话bean可以不结束事务就结束方法呢？

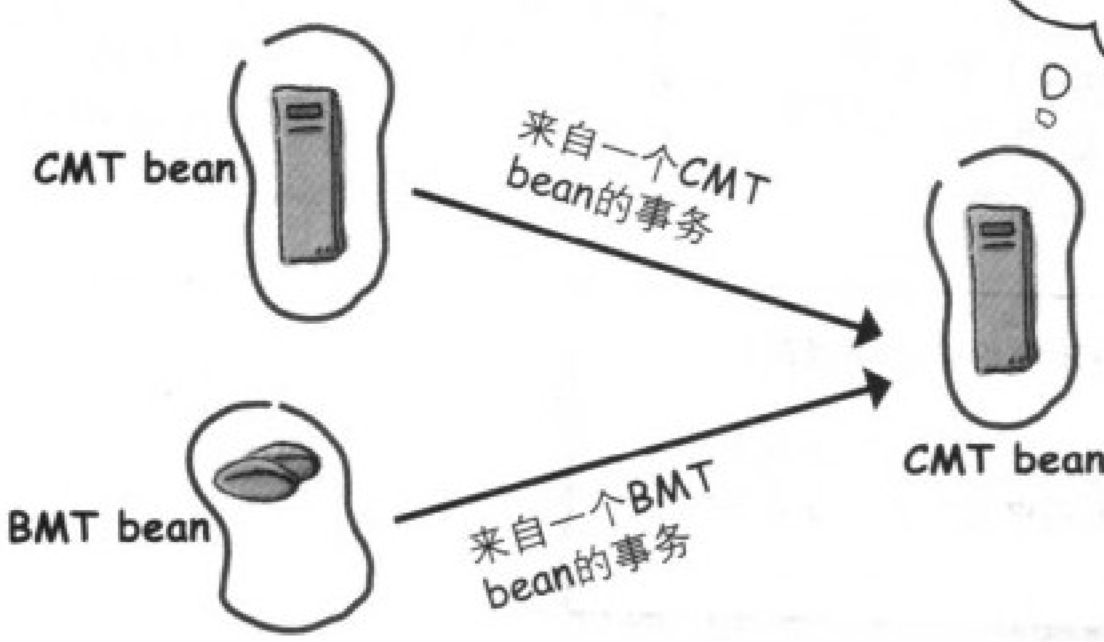
对于一个有状态，你能想到在哪种情况下可能希望这样做（事务打开时就结束方法）？

如果这样做了，可能会出什么问题？

# BMT事务是单向的：BMT事务可以传出到CMT bean，但是其他事务不能传入一个BMT bean。

BMT和CMT bean事务传播到一个CMT bean中。

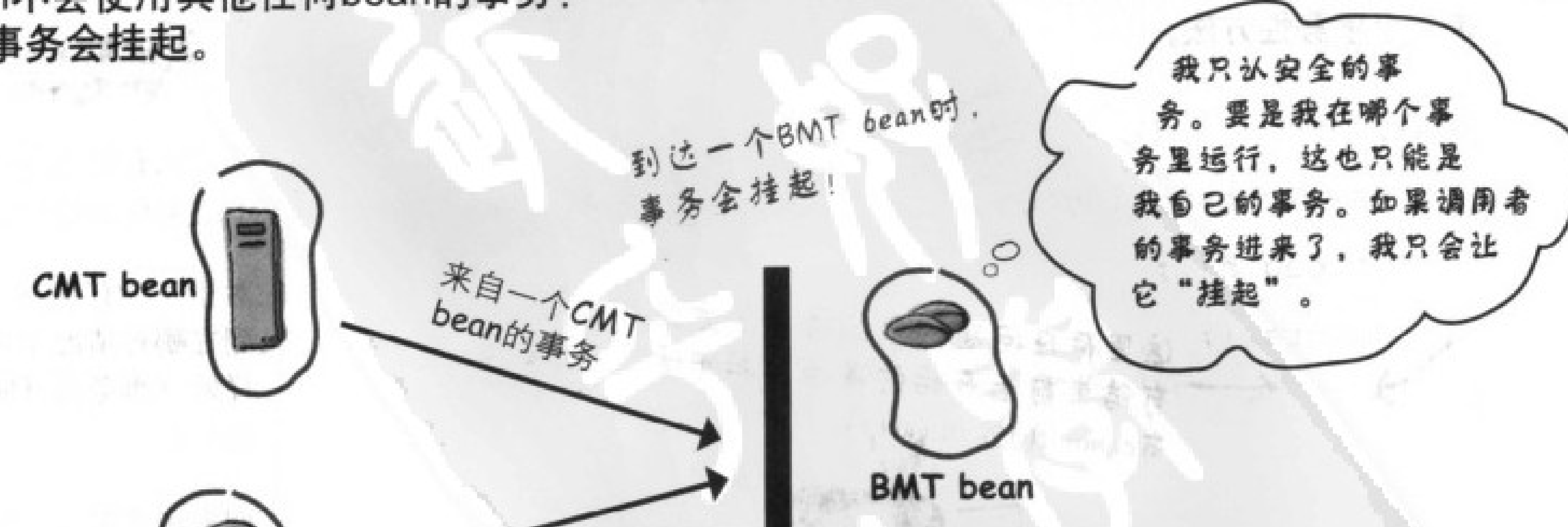



CMT bean可以在来自CMT bean的事务中运行，也可以在来自BMT bean的事务中运行。对于被调用的CMT bean来说，事务是怎么开始的（由谁开始的）没有什么分别。

---

BMT bean不会使用其他任何bean的事务！调用者的事务会挂起。



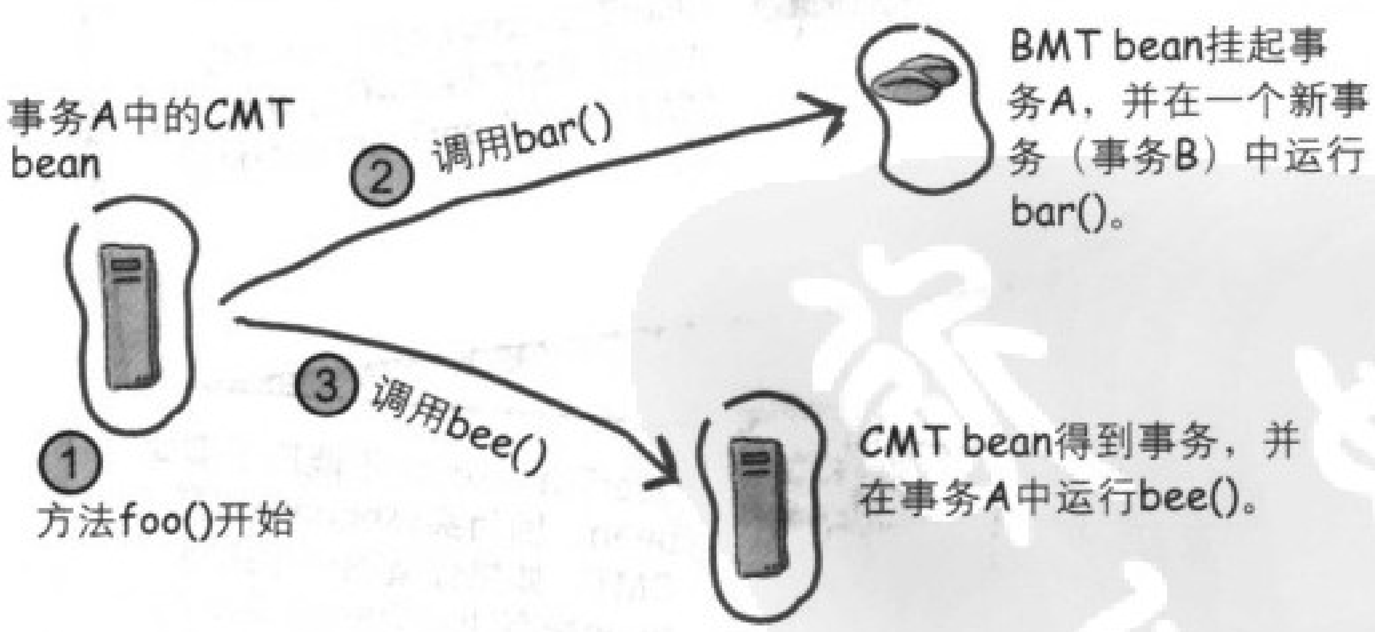

BMT bean要是在一个事务中运行，那也只会在这个bean自己创建的事务中运行。

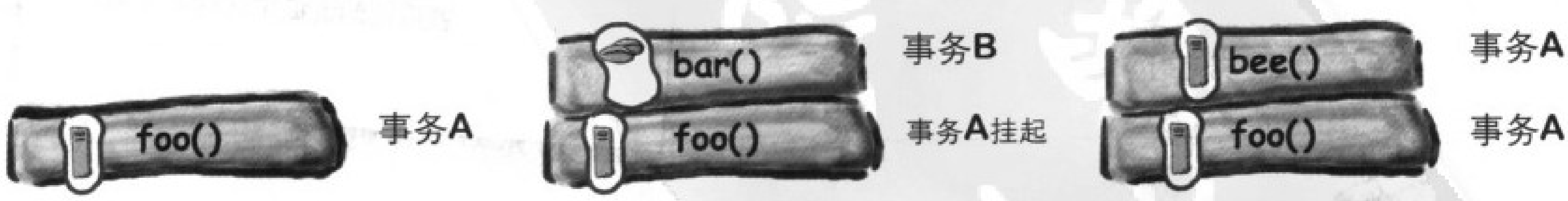
## 挂起一个事务是什么意思？

如果在一个BMT bean上调用一个方法时事务正在进行中，这个事务就会暂时挂起，等着BMT bean完成它的工作。这个工作不是调用者原来事务的一部分。一旦BMT方法（BMT bean的方法）完成，从调用栈弹出，原来的事务就会恢复，从它离开的地方继续开始。

请考虑这样一个情况：有一个CMT bean，记为bean 1，正在一个事务（事务A）中运行方法foo()，此时它调用了bean2（这是一个BMT bean）上的方法bar()。一旦 bar()结束，从栈中弹出，方法foo()就立刻调用了另一个方法bee()，不过这一次被调用的bean是另一个CMT bean（bean3）。

**一个事务挂起时，它会等待，直到从它离开的地方继续。不过，这意味着事务挂起时发生的一切不是这个原子单元的一部分。换句话说，如果挂起的事务（恢复后）无法提交，事务挂起时发生的事情并不会回滚。**




① CMT bean的方法foo()在事务A中运行。然后方法foo()调用了一个BMT bean上的方法bar()。

② 方法bar()挂起了事务A，开始了一个新事务，即事务B。方法bar()在这个新的事务B中运行，然后结束。

③ 方法bar()结束时，foo()恢复，重新使用事务A。然后它调用了bean 3上的方法bee()（这是一个CMT bean）。方法bee()在foo()的现有事务（事务A）中运行。

# UserTransaction接口

(javax.transaction.UserTransaction)

## Sharpen your pencil

UserTransaction接口有6个方法来完成BMT bean可能想做的事情。请根据所做工作的描述，写出相应的方法名（答案在最底下，倒着写的，所以先不要看下面）。

① 开始一个事务

ut.____________________

② 结束一个事务

ut.____________________

// 或

ut.____________________

③ 标记一个事务回滚

ut.____________________

④ 找出一个事务的状态

ut.____________________

⑤ 设置事务超时

ut.____________________

UserTransaction只用于BMT bean！CMT bean不能得到（或使用）一个UserTransaction引用。

UserTransaction不能用于实体bean。因为实体bean必须是CMT，如果你看到一个实体bean里有UserTransaction代码，就应该知道这个代码是非法的。

**Relax** **考试中不涉及有关事务超时的问题。**

所有事务都有一些默认的超时值，但是也可以用下面的方法修改这个默认值：

setTransactionTimeout(anIntValue)

作为一个bean开发人员，你可能只会使用默认的超时值（而不会用其他值），所以考试的时候不会考这个内容。

1. begin(). 2. commit() 或 rollback(). 3. setRollbackOnly(). 4. getStatus(). 5. setTransactionTimeout()



# setRollbackOnly()

## 事务的死亡之音

一个bean调用setRollbackOnly()时，这说明事务要完蛋了。一旦调用setRollbackOnly()，这个事务就永远也不能提交。

那么，宣布一个事务死刑是什么意思呢？这说明，这个事务绝对不能提交。好像没什么，但是要知道，这还意味着事务中的所有参与者（所有bean）都可以检查事务是否已经标记为要死掉。

记住，一个BMT bean开始的事务可能会传播到这个BMT bean对CMT bean做的方法调用中。这样，CMT bean就在BMT bean开始的事务中运行，它可能想宣布事务死刑，或者至少想知道这个事务是否已经要死了。

在一个CMT bean中，通过setRollbackOnly()可以告诉容器它不能提交这个事务。如果在业务逻辑中可以发现有一个事务会以失败告终，那就调用setRollbackOnly()。容器并不是此时结束这个事务，但是在合适的时间结束事务时，肯定不会是提交（只能是回滚）。

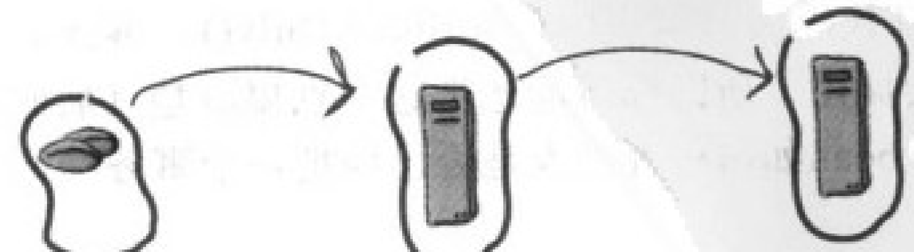
那么，事务到底什么时候结束呢？假设没有抛出任何系统异常，开始这个事务的CMT方法结束时，事务就会结束，或者对于一个BMT bean，当bean的代码调用commit()或rollback()时，事务就会结束。

真是悲哀。

调用setRollbackOnly()方法时，会设置一个标志，这就告诉其他bean，这个事务最后只有一个归宿……一个可怕的归宿。

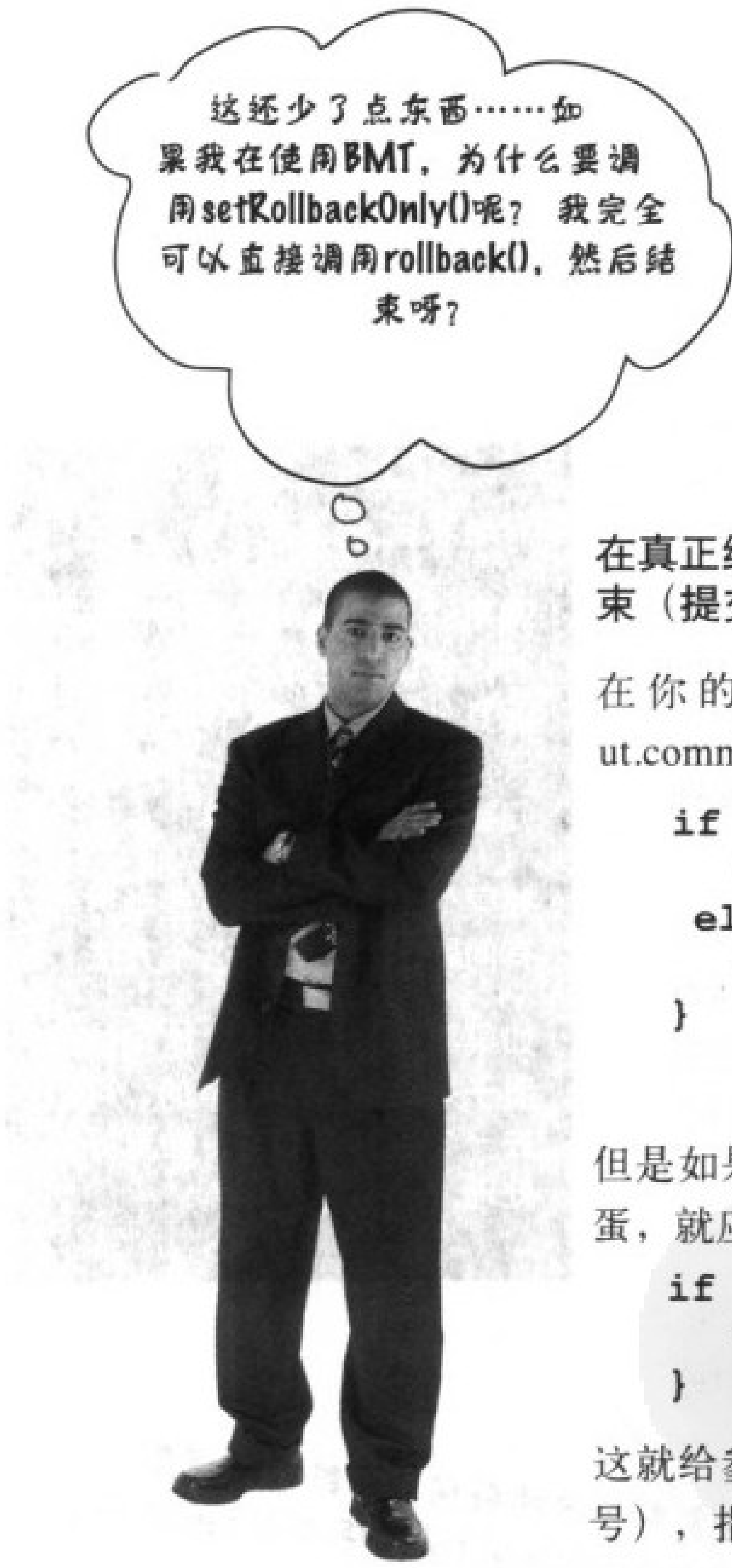
如果你知道代码中有些地方会有问题，就要调用这个方法。

如果事务肯定要死，这可能是出于好意。这是有意的，可以去一个更好的地方，做更好的处理。




这个事务中的任何bean都可以调用setRollbackOnly()以确保事务Z绝对不会提交。

如果bean B调用了setRollbackOnly()，它不会关心bean C做了什么，bean C中的事务代码永远不能提交。如果bean C能早点发现这一点（而不是做了一大堆工作之后才发现不能提交）不是更好吗？稍后将会介绍这个内容。




**在真正结束一个BMT事务之前，你可能知道这个事务应该如何结束（提交还是回滚）。**

在你的BMT代码中，可能在事务性代码的最后有一个ut.commit()或ut.rollback()。这里可能只是一个简单的if测试：

```
if (thingsLookGood) {
   ut.commit();
 else {
   ut.rollback();
}
```

一个典型示例：在事务的最后，基于某些条件提交或回滚。

但是如果在代码中前面某个地方你已经知道了这个事务肯定要完蛋，就应该调用setRollbackOnly():

```
if (thingsLookBad) {
   ut.setRollbackOnly();
}
```

这就给参与这个事务的其他人一个信号（如果他们想检查这个信号），指出这个事务已经要死了。

就算是你的BMT代码没有调用EsetRollbackOnly()，事务中的其他代码也可能会调用，所以你可能想把这一点搞清楚（后面几页我们将介绍一个bean如何检查别人是否已经把一个事务标记为回滚）。

# 两个接口里都有setRollbackOnly()

**UserTransaction**

begin()
commit()
getStatus()
rollback()
**setRollbackOnly()**
setTransactionTimeout()

UserTransaction中的所有方法仅用于BMT bean!

**EJBContext**

getCallerPrincipal()
getEJBHome()
getEJBLocalHome()
getRollbackOnly()
getUserTransaction()
isCallerInRole()
**setRollbackOnly()**

EJBContext中的setRollbackOnly()仅用于CMT bean!

UserTransaction接口中的方法只用于BMT bean；CMT bean不能使用UserTransaction中的任何方法。

EJBContext接口则不同，它同时面向BMT和CMT bean，但两个事务方法除外。

EJBContext中的setRollbackOnly()和getRollbackOnly()方法不允许BMT bean使用。

重点是：BMT bean调用UserTransaction上的setRollbackOnly()；而CMT bean调用EJBContext上的setRollbackOnly()。

**一定要知道两个setRollbackOnly()方法的规则!**

考试时很可能要考BMT和CMT bean的setRollbackOnly()如何使用。

EJBContext.setRollbackOnly()

UserTransaction.setRollbackOnly()

记住，要想同时使用这两个方法，对任何一个bean来说这都是不可能的。

CMT bean只能使用EJBContext.setRollbackOnly()。

BMT bean只能使用UserTransaction.setRollbackOnly()。

考试时可能会给出一些代码示例，让你通过查看代码来得出这个bean是BMT还是CMT。可能不会像是否有一个getUserTransaction()调用这么简单。例如，如果你看到一个EJBContext.setRollbackOnly()调用，应该知道这肯定是一个CMT bean。所以，如果这个bean后来又开始了一个UserTransaction，你就会知道这个代码是非法的。

**调用setRollbackOnly()的bean必须处在事务中!**

只有在一个事务中时才能从一个BMT bean调用setRollbackOnly()。换句话说，只是位于ut.begin()和ut.commit()或ut.rollback()之间的代码才能调用setRollbackOnly()。对于一个CMT bean，调用setRollbackOnly()的方法必须有一个适当的事务属性（后面几页将介绍这个内容）。

## getRollbackOnly()

### 生命苦短，不容bean浪费光阴

**CMT bean调用getRollbackOnly()来看它们所处的事务是不是注定会死掉。如果事务永远也不会提交，这些bean何必浪费时间去执行那么多代码呢？**

一旦bean调用了setRollbackOnly()，事务就被宣布死刑。它永远也不可能提交。但是这个事务可能还有很长的路要走，还会调用其他bean的其他许多方法，涉及到大量的代码执行。

假设你是一个bean。如果事务在调用你的方法之前就已经知道注定会死掉，但是没有人告诉你，你想想会有什么感觉？



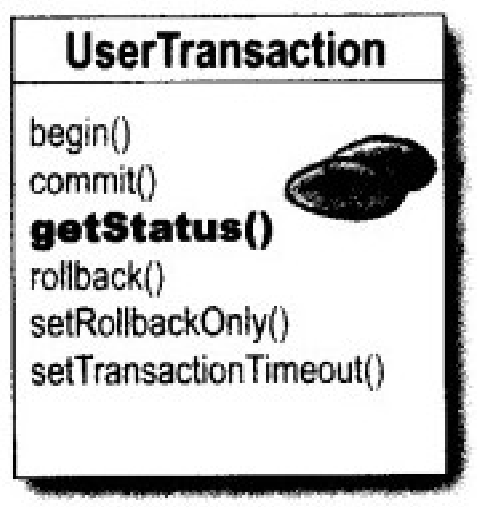

如果你是一个CMT bean，可以调用getRollbackOnly()来看看事务是否已经宣布死刑。如果确实如此，又何必费力地去做什么工作呢？

```
if (!getRollbackOnly()) {
      saveWorld();
} else {
    abandonAllHope();
}
```

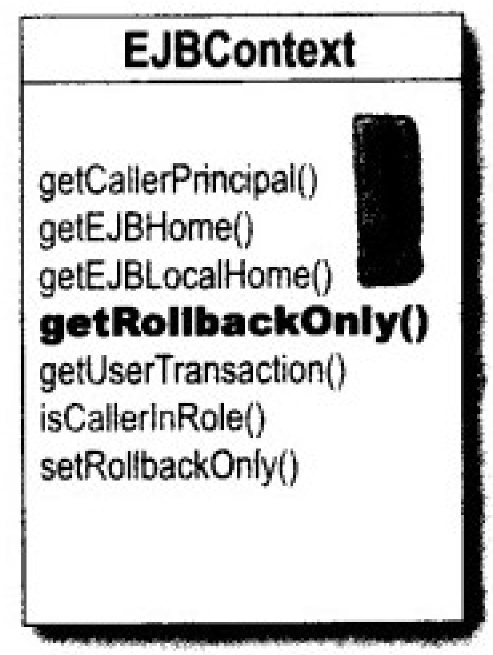



getRollbackOnly()不能用于BMT bean!只有CMT bean可以调用getRollbackOnly()。

# BMT bean使用getStatus()而不是getRollbackOnly()




UserTransaction中没有getRollbackOnly()。BMT bean要调用getStatus()。




只能CMT bean使用getRollbackOnly()。

getRollbackOnly()方法返回一个布尔值——如果方法已经标记为回滚，则返回true，如果没有要求任何人回滚，则返回false。CMT bean对于事务状态所能了解（以及需要了解）的只有这么多。

BMT bean则不同，它们还参与着事务的控制，而且可能想知道更多的东西。UserTransaction中的getStatus()方法可以告诉你想知道的任何信息，还不只如此，还能从这个方法了解到事务是怎么完成的。

**Relax 你无需记住状态常量。**

getStatus()方法返回一个表示某个常量的int，如：STATUS_ACTIVE、STATUS_COMMITTED、STATUS_COMMITTING、STATUS_THINKING_ABOUT_COMMITTING（这个常量是开玩笑的）、STATUS_MARKED_ROLLBACK、STATUS_ROLLING_BACK，还有我们个人最爱用、最常用的STATUS_UNKNOWN。

这些常量在javax.transaction.Status接口中定义（这个接口没有方法，只有一堆这样的状态常量），如果你在编写BMT代码，就会发现这些常量很有帮助。但是考试的时候不会考。你一定要知道UserTransaction中使用的是getStatus()方法，这也是BMT bean了解是否有人调用了setRollbackOnly()的惟一途径，不过要求你知道的只有这么多。

# BMT可能是一个很糟糕的想法。BMT会破坏bean重用




你能想出来这是为什么吗?

想想你在前几页学到的，特别是有关事务传播的内容（BMT的事务传播是单向的）。

**如果你编写一个BMT bean，别人将无法把你的bean包括到他们的事务当中!**

你的BMT bean形成一堵很厚的高墙，调用事务（这是指调用BMT bean的方法时所处的事务）都无法逾越。要记住，BMT bean只能在它自己创建和开始的事务中运行。如果把事务划分锁定在bean内部，这就违反了组件模型的主旨。应该记住，组件模型的妙处就在于组件可以采用新的方式进行组合，来建立bean提供者甚至从来没有想到过的新应用。部署描述文件的作用就是为应用组装人员提供一个途径，可以配置特定于某个应用的事务，而根本不用修改bean代码!

**既然使用BMT这么不好，为什么它还存在呢?**

这是因为有一些事情用CMT无法做到。但是在大多数情况下，你并不需要这些事情。

**利用BMT，可以缩小事务的作用域。**

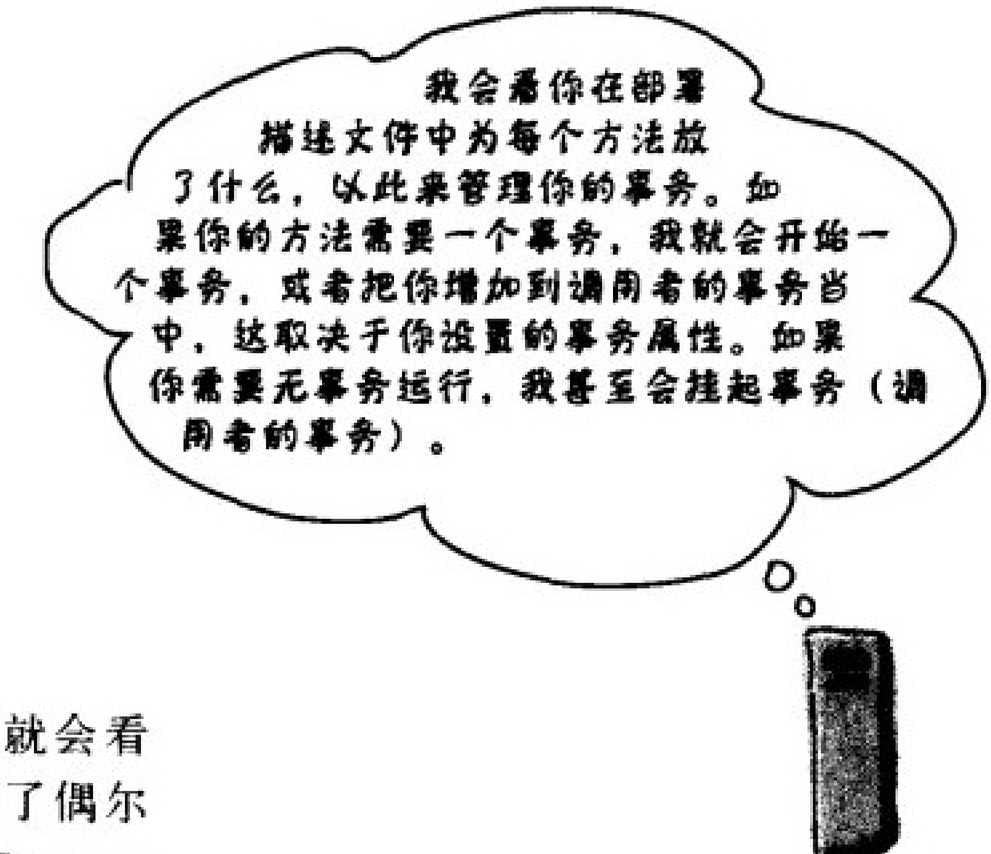
利用CMT的话，至少要在方法级而不能在更小的作用域上标记事务。在部署描述文件中，只是指出了哪个方法要有哪个事务属性（这个内容马上就会谈到），而不能指定方法的一部分有什么事务属性。不过，对于BMT，开始一个事务之后，可以在更小的作用域上将其结束。这样可以提高性能，因为事务持续的时间越长，就越有可能影响并发性。不过，由此会带来一个副作用，这就是bean的重用性会遭到破坏，所以一般来讲都是不值得的，通常还有更好的办法来改善性能……

**使用BMT，可以让一个有状态会话bean事务在客户的多个调用期间一直打开。**

利用BMT，可以在一个方法中打开一个事务（调用ut.begin()），而在这个方法结束时不结束事务（对于消息驱动bean或无状态会话bean来说，这是做不到的）。不过这通常是一个很不好的设计思想，所以不能作为采用BMT的理由。

**利用BMT，可以将事务提交状态与消息确认分离。这可以作为采用BMT的理由。**

我们在关于消息驱动bean的一章中详细介绍过这个内容，不过这里可以简单总结一下：采用CMT，只有当（而且如果）事务提交时才会发送消息确认。在某些设计中，如果消息本身有问题，就可能导致事务无法提交，而不能发送消息确认，以至于工作无法正常进行。




## 容器托管事务

前面我们已经了解了如何亲自动手划分事务，下面就会看到使用CMT是何等的容易。这实在是太容易了，除了偶尔也许需要做一个EJBContext.setRollbackOnly()或EJBContext.getRollbackOnly()调用外，你甚至不用编写任何事务性代码。

利用CMT，事务会由容器来开始和结束（通过提交或回滚），这完成取决于部署描述文件。你（从技术上讲，就是应用组装人员）要在部署描述文件中标记一些属性，仅此而已。

基本上就是这样。

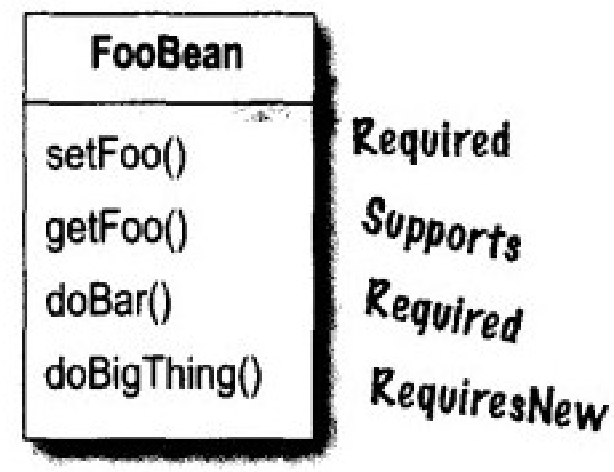
除非你很清楚这6个事务属性是什么，并且了解不同属性的组合在运行时有什么影响，否则你甚至不知道你的bean是否在一个事务中，这个事务有多大，或者你是不是创建了一种危险的情况（可能在运行时导致崩溃）。

好在只有6个事务属性。而且对于各个属性容器会有怎样的表现，这些并不复杂，很清楚也很简单。

# 属性的作用

❶ **把方法标记为以下6个事务属性之一：**

- Required
- RequiresNew
- Mandatory
- Supports
- NotSupported
- Never




❷ **调用一个方法时，容器使用这个属性来做以下5个事情中的一件：**

- 在调用者的事务中运行这个方法。

或

- 挂起调用者的事务，并开始一个新事务。

或

- 挂起调用者的事务，并且无事务运行这个方法。

或

- 抛出一个异常，因为调用者没有事务。

或

- 抛出一个异常，因为调用者有一个事务。

```
void go() {
    aFooBean.setFoo();
}
```






## 了解你的属性

**属性对行为有怎样的影响，这取决于一点：**

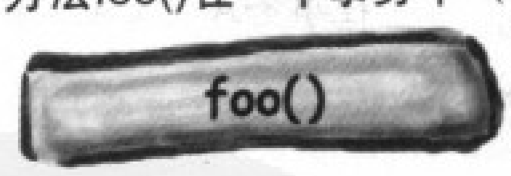
**调用方法（调用另一个方法的方法）是否在一个事务中？**

```
public void foo() {
    aBean.bar();
}
```

foo()调用另一个bean的bar()时在一个事务中（事务A）。

bar()方法标记为Required。

① 方法foo()在一个事务中（事务A）




② 方法bar()的事务属性标记为Required。

```
<method>
    <ejb-name>MyBean</ejb-name>
    <method-name>bar</method-name>
</method>
<trans-attribute>Required</trans-attribute>
```

bar()方法在foo()所在的事务中运行，因为bar()的事务属性标记为Required，而foo()有一个事务。如果foo()不在一个事务中，容器会开始一个新事务，让bar()在其中运行。

③ bar()方法在调用者的事务（事务A）中运行。



# 需要事务的事务属性

| Bar()的属性 | foo()事务状态 | 结果 |
| --- | --- | --- |
| **Required**<br>如果调用方法时已经有一个事务上下文，这个方法就会在该现有事务中运行。如果没有事务，容器会开始一个新事务。 | foo() tx A<br>在事务A（tx A）中 | bar() tx A<br>foo() tx A<br>bar()在 tx A中运行 |
| | foo() —<br>无事务 | bar() tx A<br>foo() —<br>容器为 bar()开始一个新事务(tx A) |
| **RequiresNew**<br>这个方法总会在一个新事务中运行。如果调用方法时已经有一个事务上下文，调用者的事务会挂起，直到这个方法结束才恢复。 | foo() tx A<br>在事务A（tx A）中 | bar() tx B<br>foo() tx A<br>容器挂起tx A，并为 bar()开始一个新事务 (tx B) |
| | foo() —<br>无事务 | bar() tx A<br>foo() —<br>容器为 bar()开始一个新事务(tx A) |
| **Mandatory**<br>危险！ Mandatory实际上表示"RequiresExisting"，如果调用方法时没有一个现有的事务上下文，容器会抛出一个异常！ | foo() tx A<br>在事务A（tx A）中 | bar() tx A<br>foo() tx A<br>bar()在 tx A中运行 |
| | foo() —<br>无事务 | 异常！ |

# 不需要事务的事务属性

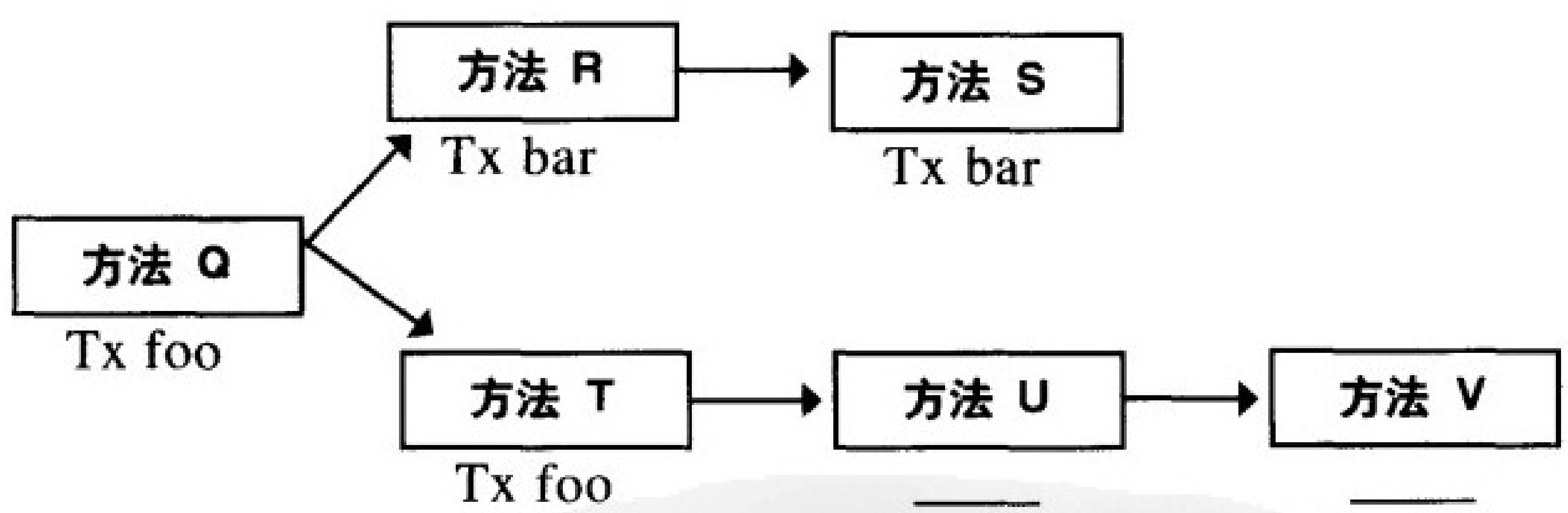
| Bar()的属性 | foo()事务状态 | 结果 |
| --- | --- | --- |
| **Supports**<br>如果调用方法时已经有一个事务上下文，这个方法会在该事务中运行。如果没有现有事务，方法会基于“未指定事务上下文”运行。 | foo() tx A ➡<br>在事务A（tx A）中 | bar() tx A<br>foo() tx A<br>bar()在 tx A中运行 |
| | foo() — ➡<br>无事务 | bar() —<br>foo() —<br>bar()基于“未指定事务上下文”运行 |
| **NotSupported**<br>如果调用方法时已经有一个事务上下文，调用者的事务会挂起。不论是否有一个现有事务，这个方法都会基于一个“未指定事务上下文”运行。 | foo() tx A ➡<br>在事务A（tx A）中 | bar() —<br>foo() tx A<br>容器挂起tx A，bar()基于一个“未指定事务上下文”运行。 |
| | foo() — ➡<br>无事务 | bar() —<br>foo() —<br>bar()基于一个“未指定事务上下文”运行。 |
| **Never**<br>Never 意味着“No Pre-Existing”。如果调用方法时已经有一个事务上下文，容器会抛出一个异常。如果原来没有事务，方法会基于一个“未指定事务上下文”运行。 | foo() tx A ➡<br>在事务A（tx A）中 | 异常！ |
| | foo() — ➡<br>无事务 | bar() —<br>foo() —<br>bar()基于一个“未指定事务上下文”运行。 |

## Sharpen your pencil

### 了解事务属性

考试要求你能够得出哪些属性的组合可以（或者会）导致一个特定的结果。可能让你看一系列方法，并且显示出这些方法在哪个事务中运行，让你得出哪些事务属性的组合会使这种情况成为可能。这种题目形式如下……

（答案在下一页的最下面。）




问题：哪两个属性组合可能带来这种结果？

1）R-Supports，S-Required，T-Mandatory，U-NotSupported，V-Never

2）R-RequiresNew，S-Required，T-Required，U-Never，V-NotSupported

3）R-RequiresNew，S-Supports，T-Supports，U-NotSupported，V-Supports

4）R-Requires，S-Mandatory，T-Mandatory，U-Supports，V-Never

5）R-RequiresNew，S-Required，T-Required，U-NotSupported，V-NotSupported

## Sharpen your pencil

要参加考试，你必须知道有关事务的一些确实很重要的规则（一般来讲，对bean开发人员也很重要），如果你现在花点时间自己得出结论会容易得多。理解要比死记好，如果非要死记硬背的话这里要记的太多了。后面几页会给出所有问题的答案，不过你要先自己做出来。

**1** **在6个事务属性中，哪一个（或者哪一些）不能由调用getRollbackOnly()或setRollbackOnly()的bean使用？**

---

**2** **哪个（或哪些）事务属性不能由消息驱动bean使用？**

（提示：要记住，消息驱动bean没有“客户”，调用onMessage()方法的是容器。）

---

**3** **你认为在什么情况下容器应该自动地回滚一个事务？**

**是bean得到一个运行时异常吗？**

**还是bean抛出一个应用异常（例如，InsufficientFundsException）？**

---

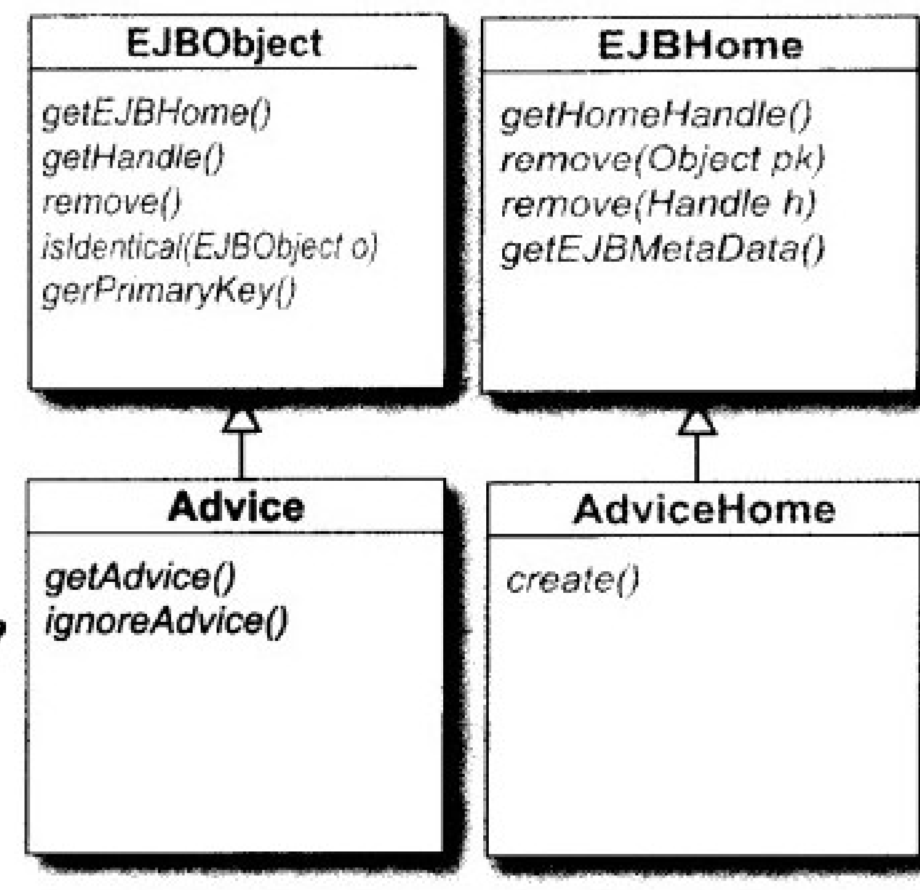
**4** **在6个事务属性当中，有3个可能很危险，其中一个还特别危险。要记住，并不是bean提供者为bean的方法指定事务属性，那么这6个事务中的哪一个可能最危险？**

有关属性的练习的答案：3和5

# 以下方法必须标记一个属性（CMT bean）

## 会话bean

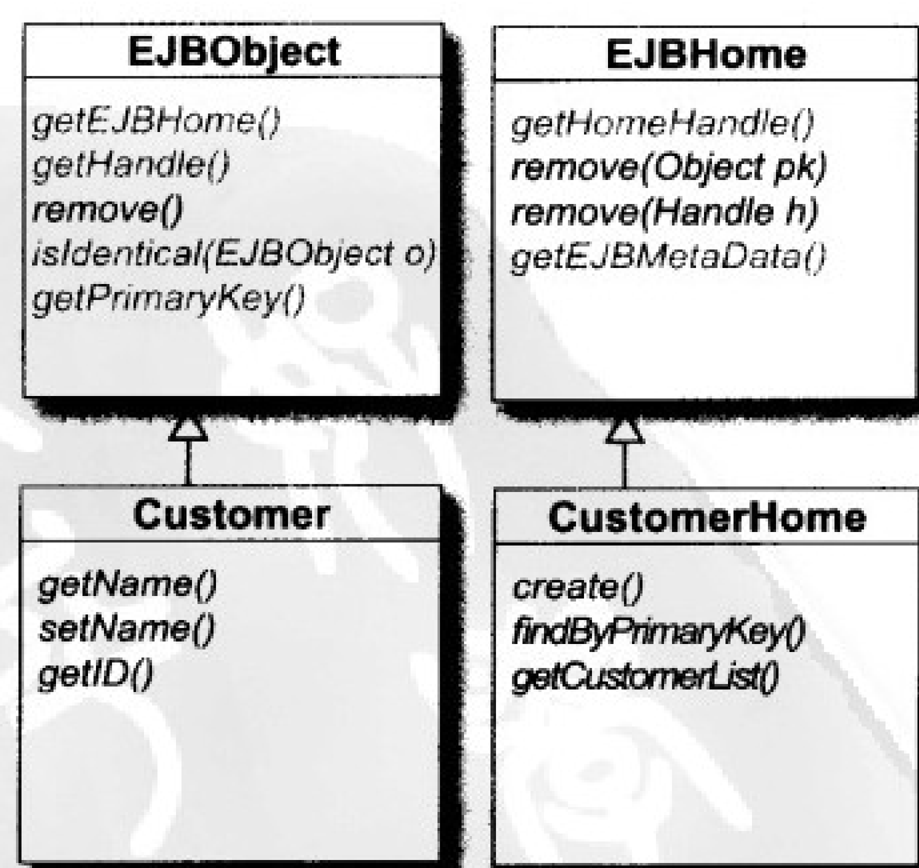
- 组件接口中的业务方法
- 客户在组件接口（EJBObject或EJB-LocalObject）中看到的其他方法都不需要属性
- home接口中的所有方法都不需要属性，包括bean提供者编写的方法，以及EJBHome或EJBLocalHome的方法。




这些方法是部署描述文件中惟一需要标记的方法。

## 实体bean

- 组件接口中的业务方法
- 除remove()外，客户在组件接口（EJBObject或EJBLocalObject）中看到的其他方法都不需要属性
- bean提供者编写的home接口方法，以及EJBHome或EJBLocalHome的remove()方法。




使用实体bean时，与会话bean相比，必须对更多的方法标记属性。要记住，create()和remove()对一个实体来说是大事（这意味着实体的真正插入和删除！）。

## 消息驱动bean

- onMessage()

消息驱动bean没有任何客户接口。

onMessage()

嗯……如果使用“Supports”会怎么样？我怎么才知道有一个事务？使用NotSupported呢？Never呢？会话bean的ejbCreate()和ejbRemove()会怎么样？如果我辞去工作，去当夏威夷考艾岛的一名冲浪教练*怎么样？



## “未指定的事务上下文”

“未指定事务上下文”这个词是EJB规范的说法，实际上就是说“你没有任何提示。我（容器）可以随心所欲做我想做的事情，你必须跟着我来！”

要参加考试，你必须知道哪些方法（在哪些情况下）可能运行在一个“未指的事务上下文”中。

- 标记为NotSupported、Never或Supports的所有CMT方法。NotSupported和Never的意思是“无事务”，但实际上，容器可以随心所欲。对于Supports，你永远不会知道事务状态（所以我们认为这是一个“哑”属性，任何人都不应使用这个属性）。
- CMT会话bean方法ejbCreate()（所有创建方法）、ejbRemove()、ejbPassivate()和ejbActivate()。我们不认为会话bean的创建方法和删除方法是客户事务的一部分（对于实体bean则有所不同）。另外要记住，如果会话bean处在一个事务中，是不会调用激活和钝化方法的。
- CMT消息驱动bean方法ejbCreate()和ejbRemove()。记住，对于一个消息驱动bean，ejbCreate()和ejbRemove()是容器调用的，容器想在池中增加bean或者想从池中删除bean时就会分别调用这些方法。这里没有调用客户事务（即调用方法时所处的客户事务），因为消息驱动bean没有真正的客户！

**这里的关键是……你不知道到底是什么！**

这里的关键是，你不能指望容器有确切的行为，因为你根本不知道它会有什么行为。作为一个bean开发人员，对此不能有一丝一毫的掉以轻心。不过你要知道哪些得到了保证，而哪些不能保证。

那么到底有没有事务？为什么是“未指定”？为什么不干脆说“根本无事务”？

规范允许容器做它想做的事情。规范提出了许多选择，从全部方法都无事务执行，到把多个对资源管理器的调用合并到一个事务中，这些做法都是允许的。

*对于最后一个问题，我们认为当滑雪教练比教冲浪更好一些。不仅同样有意思，而且可以脱开那些橡胶玩艺，另外还不会碰上鲨鱼。

# 牢记这些内容

这些东西都可能会考到。不过要记住，不会只是问你一道简单的是非题，如判断“可以从一个事务属性为NotSupported的方法调用getRollbackOnly()方法”是否正确（当然，这个问题的答案是错的）。你会看到更灵活的题目，比如让你看bean代码和bean的部署描述文件，要求你确定它们能不能正常合作。

**getRollbackOnly()必须从处在事务中的一个bean调用！**

你已经知道了，getRollbackOnly()只能由CMT bean调用，而且这个方法只是EJBContext中有，UserTransaction接口中是没有的（还记得吗？BMT bean可以调用UserTransaction上的getStatus()，但不能调用getRollbackOnly()）。

不过，要让getRollbackOnly()工作，这个方法必须在一个事务中，这说明事务属性只能使用：

* Required
* RequiresNew
* Mandatory

**消息驱动bean只能使用两个事务属性：*Requires*和*NotSupported*。**

想想看。消息驱动bean没有调用客户！是容器来调用onMessage()方法，所以使用其他事务属性是没有意义的。比如说RequiresNew，这是没有用的，因为不存在已有的事务。Mandatory呢？这样每次都会出问题（这里“出问题”的含义是抛出一个异常），因为Mandatory意思是“必须已经有”。Never呢？这是当然的了，本来肯定就是无事务的，再指定这个属性就太多余了。Supports也很傻。对于onMessage()只能说两件事：

**是的，我想开始一个事务（Required）**

或

**不，我不想要一个事务（NotSupported）**

**记住：只有系统异常才会导致自动回滚！**

我们会在有关异常的一章详细讨论这个问题，不过现在要注意的是，应用异常不会自动地导致一个回滚。应用异常是bean接口上声明的任何受查异常，除非它是一个RemoteException（RemoteException属于特殊的一类）。

假设你在考试中看到一些代码，显示出bean抛出一个应用异常（比如，一个FinderException）。不要说这个事务会回滚！如果你（bean提供者）在逻辑中认为这种情况太危险，不能让事务继续，就必须调用setRollbackOnly()。只有抛出系统异常时（如EJBException或其他任何运行时异常），容器才会自动回滚。



（我们看到一个BMT bean和一个CMT bean正在酒吧激烈争论。）

BMT：你真差劲。

CMT：这叫聪明。看你怎么说了。你说差劲是什么意思？是说我一个人差劲，还是所有CMT bean都差劲呢？

BMT：当然是你们全部。懦夫。是不是水平不够，没办法处理自己的事务，所以你们把事务都留给容器来处理。你们可能还害怕垃圾回收器吧。

CMT：懦夫？你敢说你是自己管理事务的？还不是容器负责！对你对我都一样。

BMT：不是这样的！是我来开始事务，由我决定是提交还是回滚，还有什么时候提交和回滚，这有容器什么事？我是说，容器确实必须把事务给我，但是在此之后，就是我负责了。

CMT：不对，不是你负责。

BMT：当然是我。

CMT：你做的只不过就是划分事务的边界。你只要说从哪开始，到哪结束。不就是这些吗？

BMT：哦，你忘了，我还能回滚呢。

CMT：我也能。

BMT：不，你不能。

CMT：我能。那你认为setRollbackOnly()是干什么的？

BMT：你又不能调用那个方法。那是在UserTransaction接口里的，你们这些CMT bean没这个本事。

CMT：你到底是不是个bean。你不知道EJBContext也有一个setRollbackOnly()方法吗，那就是专门给CMT bean的。

BMT：哦，我忘了。不过，那又怎样？你还是不能控制你的事务边界。

CMT：但那又有什么关系呢？我的方法都标记好了我想怎么应用事务，这和控制事务边界有什么区别呢？

BMT：拜托!你的事务至少必须有一个方法那么长！我可以任意安排我的事务边界，粒度可以比整个方法更细。

CMT：为什么要那么细呢？

BMT：问得好。为了让你听明白，我来一点一点告诉你：事务会破坏并发性。这说明，事务会影响性能和可扩展性。而且这意味着……

CMT：[打断]好了，好了，这些我都知道。但是如果没有事务，你甚至不能运行你的业务。

BMT：嗯，我又没说不使用事务，我只是说尽可能地让事务短些。这就好像同步整个方法和建立一个同步块之间的区别一样。

CMT：我看不出这有什么重要，不过，就算你说的有理吧。不，先等等，如果你的方法那么大，这说明你的设计可能不好，所以，如果这就是你们BMT的惟一优点……

BMT：不仅如此。有些事情你们做不了，可我们能做。

CMT：比方说？

BMT：比如在一个有状态bean方法里打开一个事务，而在这个bean的另外某个方法里关闭这个事务。

CMT：哦，这样难道不会毁了你的性能吗？你不知道这样会破坏你的可扩展性吗？这还会让有状态bean没办法钝化！你很危险，有可能让事务一直打开着。而且你怎么保证开始事务的那个方法一定会在结束事务的方法之前调用呢？喂，你有可能在调用begin()之前先调用了commit()或rollback()！

BMT：是这样，但是如果确实有人要这么做，那除了选择BMT，就没有别的办法。

CMT：不过，几乎所有人都不需要那么做！或者说，所有人都不应该那么做。哦，对了，如果有这样让人难以置信的不太可能的情况发生，如果开发人员确实需要这么做，不能不承认，你是惟一的办法。

不过，我看不出还有什么使用BMT的理由了，特别是，BMT会破坏整个可重用组件。你看你多自以为是！你从来不在别人的事务里运行！只能在你自己的事务里。说什么“和别人玩不好……”

BMT：嗯，可能我就是要用于一些特殊情况，不过下一次我们得谈谈怎么避免有问题的消息驱动bean消息。这方面只有我能行。

CMT：对，在有关消息驱动bean的一章我们还要见面。回头见。

# 在部署描述文件中标记事务

所有bean都必须指出它们在使用bean托管事务还是容器托管事务。对于BMT bean，在部署描述文件中只需指定这一点。

但对于CMT bean，这还只是个开始。







**与事务相关的部署描述文件信息放在两个地方：**

① CMT和BMT bean都适用：

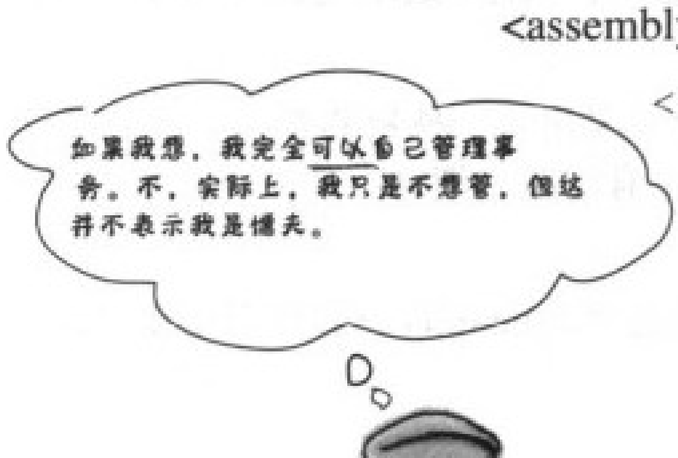
<enterprise-beans>部分中的<transaction-type>元素：

```
<enterprise-beans>
   <session>
     ...
      <transaction-type>Container</transaction-type>
   </session>
</enterprise-beans>
```

对于BMT，这个类型是bean

② 仅用于CMT bean：

<assembly-descriptor>部分中的<container-transaction>元素：




```
<assembly-descriptor>
  <container-transaction>
     <method>
        <ejb-name>MyBean</ejb-name>
        <method-name>bar</method-name>
     </method>
     <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

# CMT的部署描述文件示例

### <enterprise-beans>元素中

```
<enterprise-beans>

  <session>
    <display-name>AdviceBean</display-name>
    <ejb-name>AdviceBean</ejb-name>
    <home>headfirst.AdviceHome</home>
    <remote>headfirst.Advice</remote>
    <ejb-class>headfirst.AdviceBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <security-identity>
      <use-caller-identity></use-caller-identity>
    </security-identity>
  </session>

</enterprise-beans>
```

对于BMT，你要说"Bean"而不是"Container"

### <assembly-descriptor>元素中

```
<assembly-descriptor>
  <container-transaction>
     <method>
        <ejb-name>MyOtherBean</ejb-name>
        <method-name>foo</method-name>
     </method>
     <trans-attribute>RequiresNew</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

有多个<method>元素（下一页会分析这些元素）

通过列出一个特定属性（RequiresNew, Supports等等），并把它放在有该属性的所有方法中，以此来指定属性。你可能认为，你要先指定一个方法，然后再给这个方法一个属性（因为在这个例子中看上去就是这样），但是实际工作并非如此。下一页将详细说明……

# CMT的更多部署描述文件例子

## 选择 1: 通配符

用通配符表明特定bean中的所有方法有<trans-attribute>标记中的属性。

```
<container-transaction>
    <method>
        <ejb-name>BigBean</ejb-name>
        <method-name> * </method-name>
    </method>
    <trans-attribute>RequiresNew</trans-attribute>
</container-transaction>
```

指定一个方法要给定bean名和方法名。不过，这里我们使用了一个通配符(*)，它表示"BigBean的所有方法都有RequiresNew属性"。

## 选择 2: 单独命名方法

在ejb-jar中指定每个CMT bean的每个方法（除非使用通配符表明指定bean类的所有方法都有指定的属性）。

```
<container-transaction>
    <method>
        <ejb-name>BigBean</ejb-name>
        <method-name>foo</method-name>
    </method>
    <method>
        <ejb-name>TinyBean</ejb-name>
        <method-name>go</method-name>
    </method>
    <method>
        <ejb-name>MyBean</ejb-name>
        <method-name>bar</method-name>
    </method>
    <trans-attribute>RequiresNew</trans-attribute>
</container-transaction>
<container-transaction>
    <method>
        <ejb-name>BigBean</ejb-name>
        <method-name>doStuff</method-name>
    </method>
    <trans-attribute>Requires</trans-attribute>
</container-transaction>
```

这里有3个不同的方法，分别在3个不同的bean中，它们都有RequiresNew属性。

这是一个不同的事务属性——Requires，所以这里列出了使用Requires的方法。注意，我们指定的bean是前一个<container-transaction>元素中使用的同一个bean。




**别担心。**

你的设计有可能把一个重载方法的所有版本都以同样的方式处理（在事务方面）。在迄今为止看到的例子中，一个方法名就表示了该方法的所有重载版本。

但是，万一确实需要区别不同的重载方法，也有一种办法来指定参数。这个可选的**<method-params>**标记形式如下：

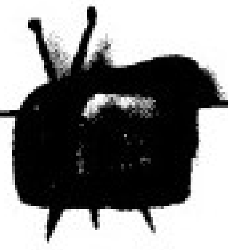
```
<container-transaction>
    <method>
      <ejb-name>ShoppingBean</ejb-name>
      <method-name>addItem</method-name>
      <method-params>
        <method-param>java.lang.String</method-param>
        <method-param>int</method-param>
      </method-params>
    </method>
    <trans-attribute>RequiresNew</trans-attribute>
</container-transaction>
<container-transaction>
    <method>
      <ejb-name>ShoppingBean</ejb-name>
      <method-name>addItem</method-name>
      <method-params>
        <method-param>int</method-param>
      </method-params>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>
```

这里有一个重载方法 addItem，其中一个版本取一个String和一个int为参数，另一个版本只取一个int参数。第一个重载方法使用RequiresNew属性，第二个版本使用Required。所以必须指定在每个部分里说的是哪一个重载版本。为此，可以使用可选的<method-params>和<method-param>标记来做到。

**事务属性是应用组装的一部分，而不是bean开发的一部分。**

**这说明属性要在部署描述文件的组装部分指定，而不是在bean部分指定。**

**在bean部分，有关事务惟一要说的就是使用容器托管事务还是bean托管事务。**



**不错，你确实需要知道有关事务的标记。**

不要求你记住整个部署描述文件中的每一个标记，但是确实需要知道如何指定事务属性，包括如何使用通配符（*）来表示一个bean的所有方法，另外还要知道有一种方法可以为重载方法提供不同的事务属性。

一定要知道，事务属性不是在部署描述文件的<enterprise-beans>部分中指定的，而是在<assembly-descriptor>部分中指定。想想看为什么这样……bean提供者必须（在部署描述文件中）指出bean到底使用BMT还是CMT，但仅此而已。应用组装人员随后要准确地指定，对于bean的这个特定部署，应当如何处理事务。

---

**问：** 可以把通配符与特定的方法名结合吗？我的意思是说，如果我希望除了一个方法之外，让bean的所有方法都使用RequiresNew，只有那个方法使用NotSupported，会怎么样呢？

**答：** 没问题。使用通配符就像是说“如果没有在部署描述文件中另外指定，所有方法都使用这个属性”。所以，尽可以对部署描述文件中的RequiresNew部分（使用RequiresNew属性的方法）使用通配符，而指定NotSupported属性时给出具体的方法名。

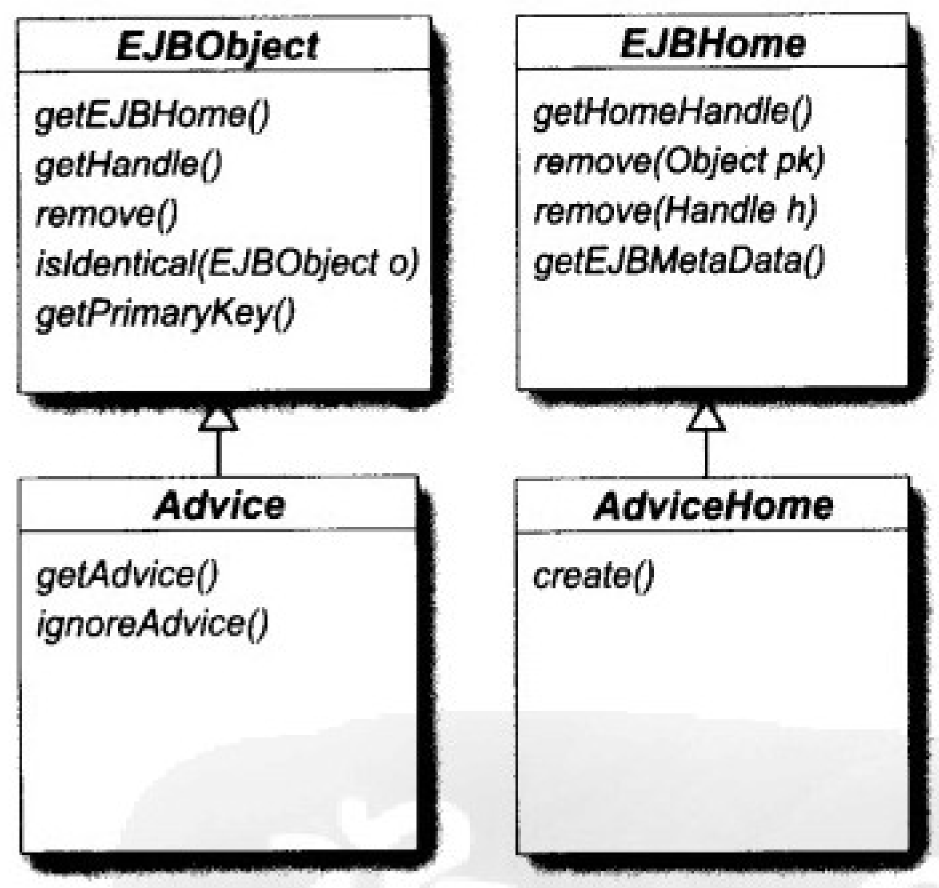
```
<container-transaction>
   <method>
      <ejb-name>BigBean</ejb-name>
      <method-name> * </method-name>
   </method>
   <trans-attribute>RequiresNew</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>BigBean</ejb-name>
      <method-name>useOldDatabase</method-name>
   </method>
   <trans-attribute>NotSupported</trans-attribute>
</container-transaction>
```

## Sharpen your pencil

### 哪些方法需要事务属性?

这个题目要求把你记住的和考虑过的结合起来。请看以下的接口，一个对应会话bean，另一个对应实体bean，得出bean使用CMT时哪些方法必须有事务属性。可以划勾、画圈、画线强调。但是不要仅把这一章的内容简单地记下来……一定要好好思考。

会话bean



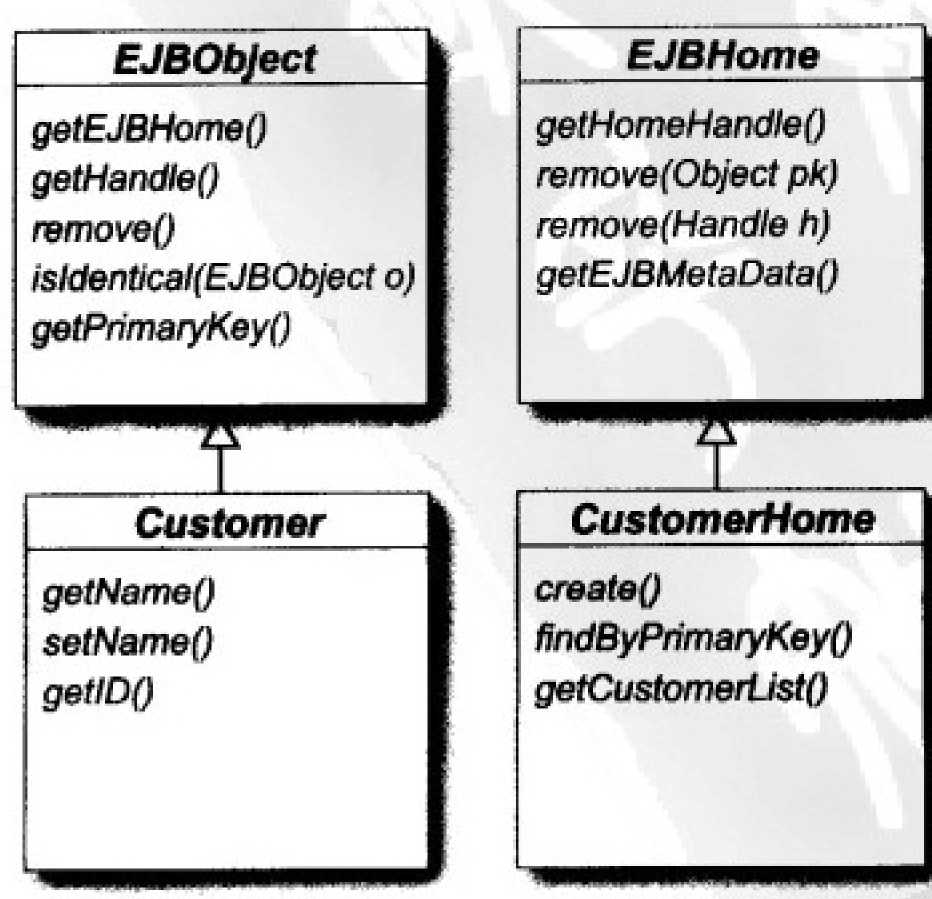

这里有一个建议，对于每个方法，请自问下面这些问题：

* 有没有理由说这个方法必须在一个事务中？如果它不在事务中，会不会有风险？

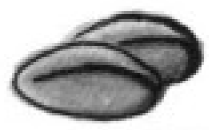
* 有没有理由说这个方法不应在一个事务中？

* 有没有理由说这个方法的事务行为应当完全由容器决定……因为它与客户要做什么没有什么关系？

实体bean



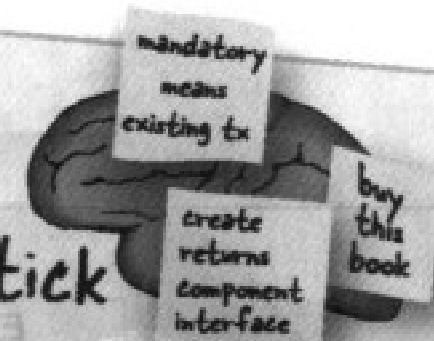
# bean托管划分小结



## bean托管(BMT)

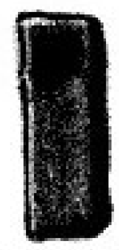
- 无状态bean和有状态会话bean都可以使用。
- 消息驱动bean可以使用。
- 实体bean不能使用。
- 可以用于缩小事务的作用域，这有助于提高性能。
- 可以用于保持事务在对一个有状态会话bean的多个调用期间一直打开。
- 可以由消息驱动bean用于确认一个消失，即使事务是回滚结束的。
- 消息驱动bean必须在onMessage()的最后结束事务。
- 无状态会话bean必须在开始事务的业务方法结束前结束这个事务。
- BMT bean在结束前一个事务之前不能开始另一个事务。（要记住，在EJB中没有嵌套事务的概念！）
- BMT bean不能使用EJBContext的getRollbackOnly()和setRollbackOnly()方法。
- BMT bean可以在UserTransaction上调用setRollbackOnly()方法。
- bean从bean的EJBContext得到一个UserTransaction。
- BMT bean中事务的传播是单向的：BMT bean开始的事务可以向外传播到其他的bean，但任何事务都不能传播到使用BMT的bean中。
- UserTransaction接口没有getRollbackOnly()方法，但是BMT bean可以调用getStatus()来得出事务中是否有人调用了setRollbackOnly()。
- getStatus()方法返回一个表示一个常量的int，其中之一是STATUS_MARKED_ROLLBACK。
- 使用BMT的会话bean不能实现SessionSynchronization。

**Make it Stick**




**无论哪里来的事务，**
**CMT bean都能运行；**
**但BMT bean不同，**
**只能运行自己的事务。**

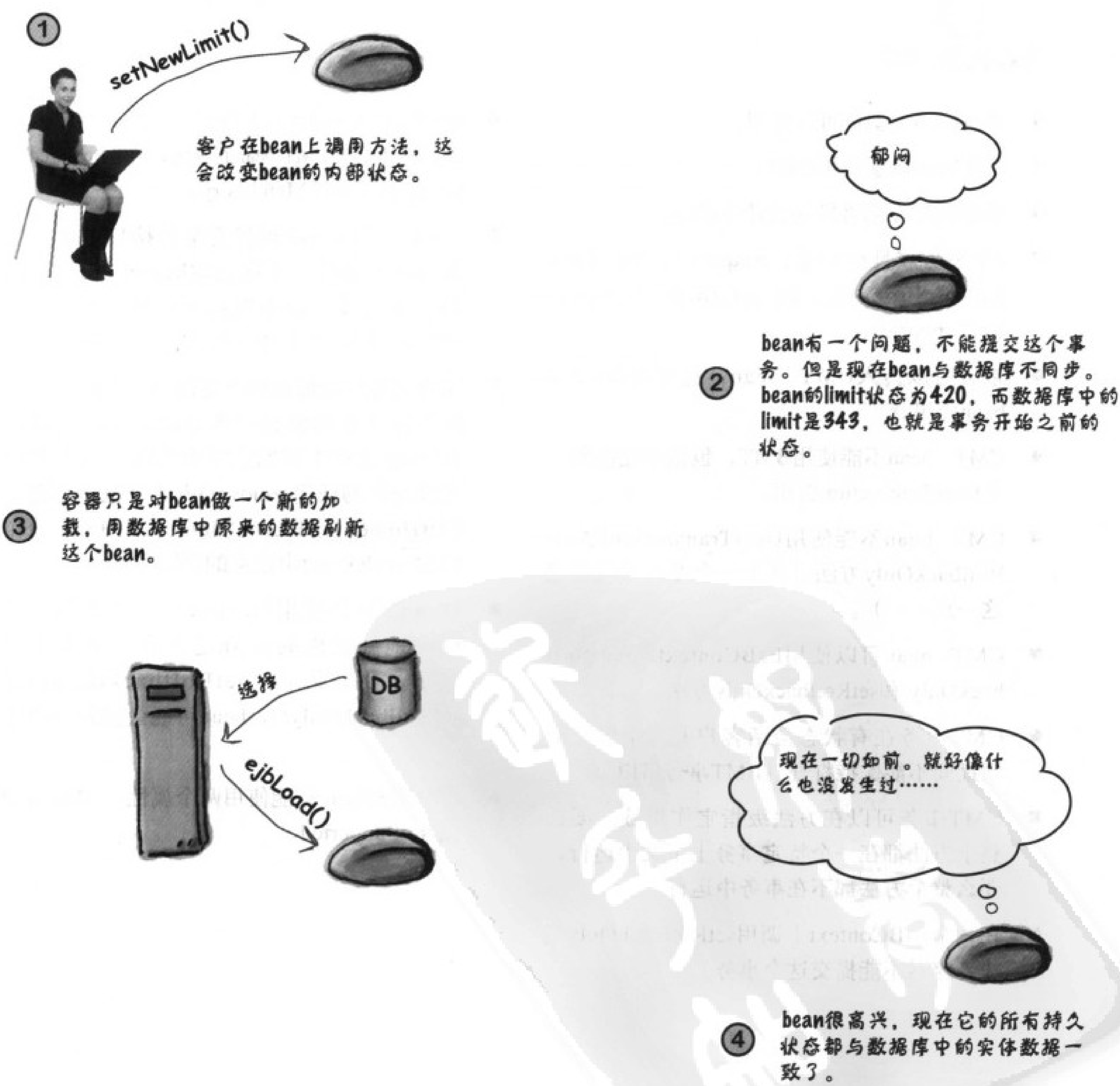
确实，这个小帖士写得不怎么样。你为什么不自己试试呢？这种小帖士能帮助记忆，不过如果你自己编一个，可能记得更牢。



## 容器托管(CMT)

- 所有bean类型都可以使用。
- 实体bean必须使用CMT。
- 事务属性在部署描述文件中指定。
- 6个事务属性分别是：Required、Requires-New、Supports、Mandatory、Never和NotSupported。
- 事务可以传入CMT bean，也可以从CMT bean传出。
- CMT bean不能使用BMT，包括不能得到一个UserTransaction引用。
- CMT bean不能使用UserTransaction的set-RollbackOnly方法（从上一个要点可以得出这一点……）。
- CMT bean可以使用EJBContext的getRoll-backOnly 和setRollbackOnly方法。
- CMT事务在有状态会话客户的多个方法调用期间不能保持打开（BMT事务可以）。
- CMT事务可以在方法级指定作用域。要么整个方法都在一个特定事务上下文中运行，要么整个方法都不在事务中运行。
- 在一个EJBContext上调用setRollbackOnly意味着容器不能提交这个事务。
- 调用setRollbackOnly()或getRoll-backOnly()的CMT bean的属性必须是Required、RequiresNew或Mandatory。
- CMT会话bean必须指定组件接口中所有业务方法的属性：不能指定home中任何方法的事务属性，也不能指定EJBObject（或EJBLocalObject）中任何方法的事务属性。
- 所有实体bean都必须指定以下方法的属性：组件接口中的所有业务方法，bean提供者在home接口中声明的所有方法，以及客户可以访问的任何remove()方法（也就是说，EJBHome、EJBLocalHome或EJBObject和EJBLocalObject中定义的删除方法）。
- 你可能不想使用Supports，因为你不能确切地知道你的bean是否在一个事务中运行。如果调用了getRollbackOnly()或setRollbackOnly()，bean可能会得到一个异常。
- 消息驱动bean只能使用两个属性：Required或NotSupported。

# 即使事务回滚，实体bean也可以利用ejbLoad()保持同步。



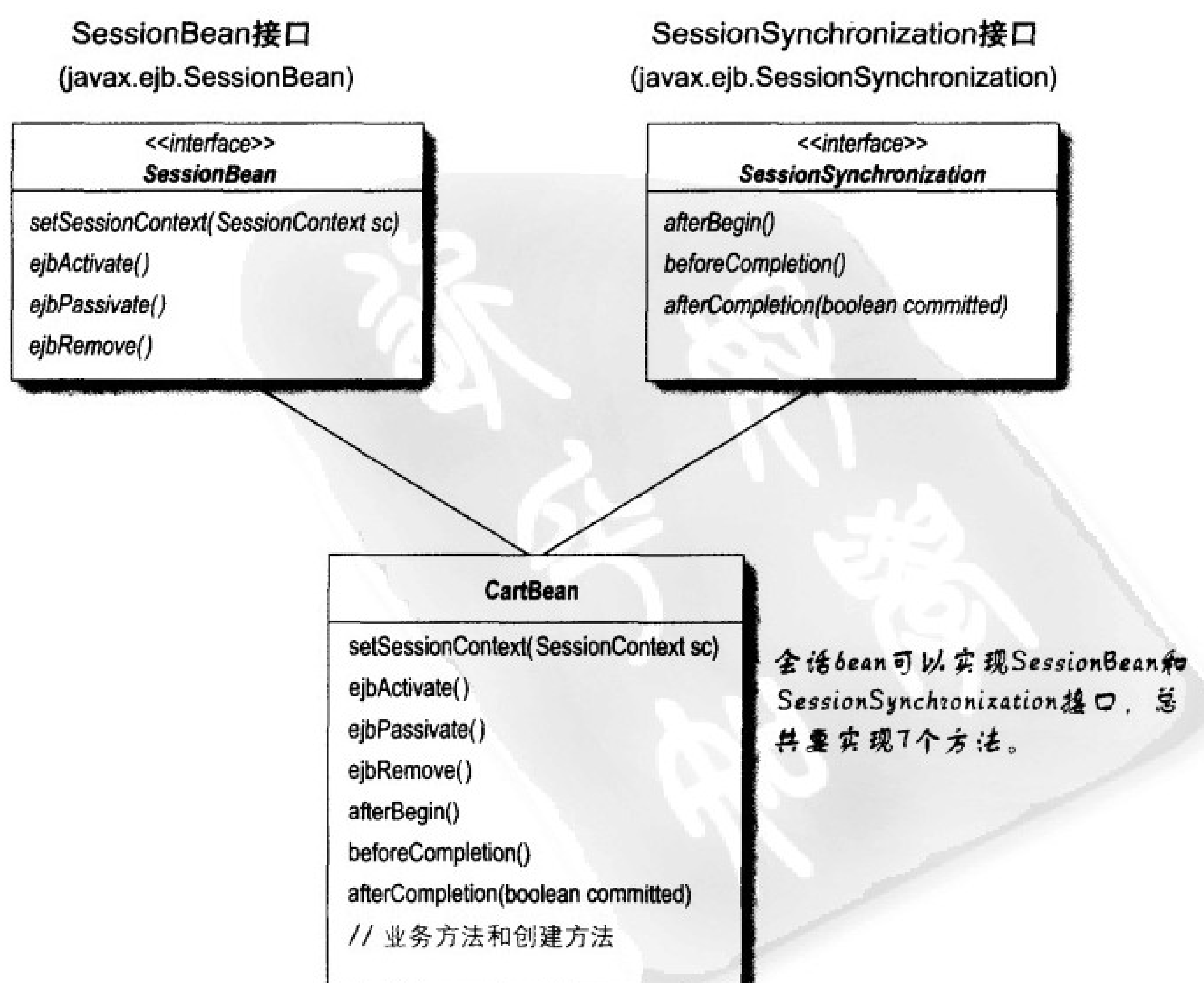
# 会话同步

## 因为会话bean没有ejbLoad和ejbStore

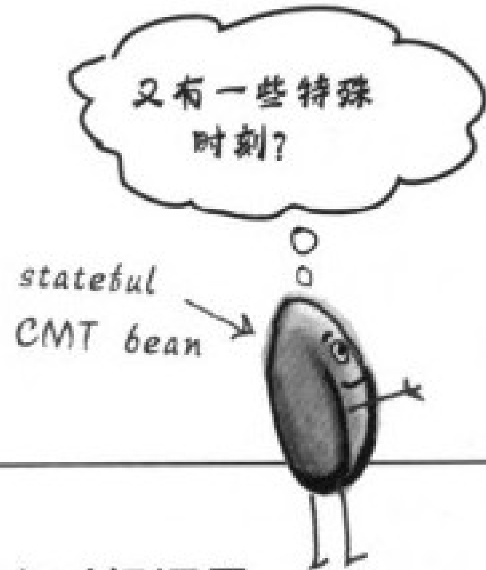
关键是：

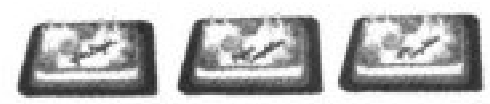
实体bean有ejbLoad()和ejbStore()，可以告诉它什么时候与数据库同步。如果事务要提交，就会调用ejbStore()让bean有最后一次机会得到它当前的持久状态，以便写到数据库中。如果事务不提交，bean只是得到另一个ejbLoad()调用，返回到事务之前的状态，这样一切仍正常。

但是会话bean没有这么幸运。它没有ejbLoad()或ejbStore()来告诉它什么时候自行与数据库同步。但是，如果你的会话bean实现了SessionSynchronization，这会让会话bean有了3个新的容器回调，可以用来通知bean在它的事务生涯中出现了3个特殊的时刻：事务开始的时候、事务要结束的时候、事务结束的时候。



# SessionSynchronization“特殊时刻”









| | afterBegin() | beforeCompletion() | afterCompletion() |
|---|---|---|---|
| 什么时候调用 | 在有状态bean的事务之初，而且是将在事务中运行的业务方法得到调用之前。 | 事务结束之前，在业务方法完成之后。 | 事务提交或回滚结束之后。方法传递一个布尔参数来告诉你事务是否提交。 |
| 与哪个实体bean方法最相似 | ejbLoad() | ejbStore() | 实体bean生命周期中没有与它对应的方法（因为没有这个需要）。 |
| 此时要做什么 | 从数据库加载数据，由于知道数据库资源现在是事务的一部分，所以可以把数据缓存在bean中,以便接下来在事务中使用。 | 在事务提交而且数据库上的锁释放之前，这是最后一次更新数据库的机会。 | 得出事务完成得怎么样，尽一切可能保持与数据库的同步。例如，如果把新修改的数据缓存在临时变量中，现在可以把这些临时值赋给持久状态变量。或者，如果事务回滚，可以用数据库中的数据重置一些值。 |
| 此时可以做的bean工作 | 调用SessionContext上的方法：得到home，得到EJB对象，得到调用者安全信息，要求事务回滚，检查回滚状态。<br><br>可以访问：特殊的JNDI上下文，资源管理器，其他bean。你在一个事务中，所以一切都很安全！ | 还是在一个事务中……所以与afterBegin()中能做的bean工作完全一样。 | 调用SessionContext上的方法：得到home，得到EJB对象，得到调用者安全信息，但不能做任何与事务相关的事情，因为你已经不在一个事务中了！只能访问特殊的JNDI上下文。现在访问资源管理器或其他bean是不安全的。 |

**无状态会话bean不能实现SessionSynchronization!**

只有有状态会话bean才能实现SessionSynchronization，因为无状态会话bean一旦方法结束就不能再维持一个事务了。

**SessionSynchronization只适用于CMT bean!**

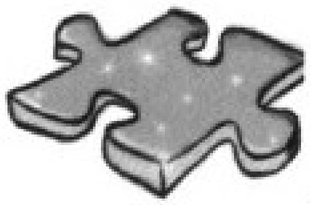
BMT bean使用SessionSynchronization有意义吗？没有。要记住，SessionSynchronization的关键是再给bean 3个"特殊时刻"回调方法，以便bean知道事务什么时候开始和结束，以及如何结束。不过请想想看……BMT bean完全知道一个事务何时开始和结束，因为正是由这个bean来划分事务边界，正是这个bean来宣布事物何时开始与结束。除了开始事务的bean以外，谁也不能结束这个事务，所以没有什么新鲜玩艺需要告诉这个bean。

## there are no Dumb Questions

**问：** **如果要这么麻烦地同步，为什么不直接使用一个实体bean呢？**

**答：** 因为你的bean可能表示一个过程，而不是一个实体。并不是涉及数据库数据的就是实体！如果你的会话bean表示一个过程，比如说一个购物过程，但是它还使用一个数据库，可能就希望在更新数据库中的信息之前先等着事务完成。

另一方面，如果只是为了避免实体bean而使用会话bean来担当"实体bean"，这样做也很傻。如果需要一个实体，那就建立一个实体bean，这可以得到容器必须提供的所有好处，比如说与持久存储库自动同步。但是如果你的bean是一个过程，还需要了解事务如何进行，就可以使用SessionSynchronization（只要这是一个有状态的 CMT bean）。



# 池填空



你的任务是从池中取出方法，填在右侧代码段的横线上。一个方法只能用一次。

你的目标是在BMT bean中建立一个方法，它一共会创建5个事务。不过别着急，还有呢！这个方法中要两次临时挂起一个事务（究竟是同一个事务被挂起两次，还是两个不同的事务被挂起，这一点你要自己考虑，我们不会告诉你）。名字里有“cmt”的方法都来自一个CMT bean，而且方法的事务属性从方法名中可以看出来。

假设变量“c”和“ut”都适当地初始化为实例变量。

哦，对了，不能抛出任何与事务有关的异常！

（答案在下一页，所以先不要看，除非你觉得直接偷看答案无所谓。）

```
void bmtMethodWithTransactions () {

  ____________________

  c.cmtRequiresNew();

  ____________________

  ____________________

  ____________________

  ____________________

  ____________________

  ____________________

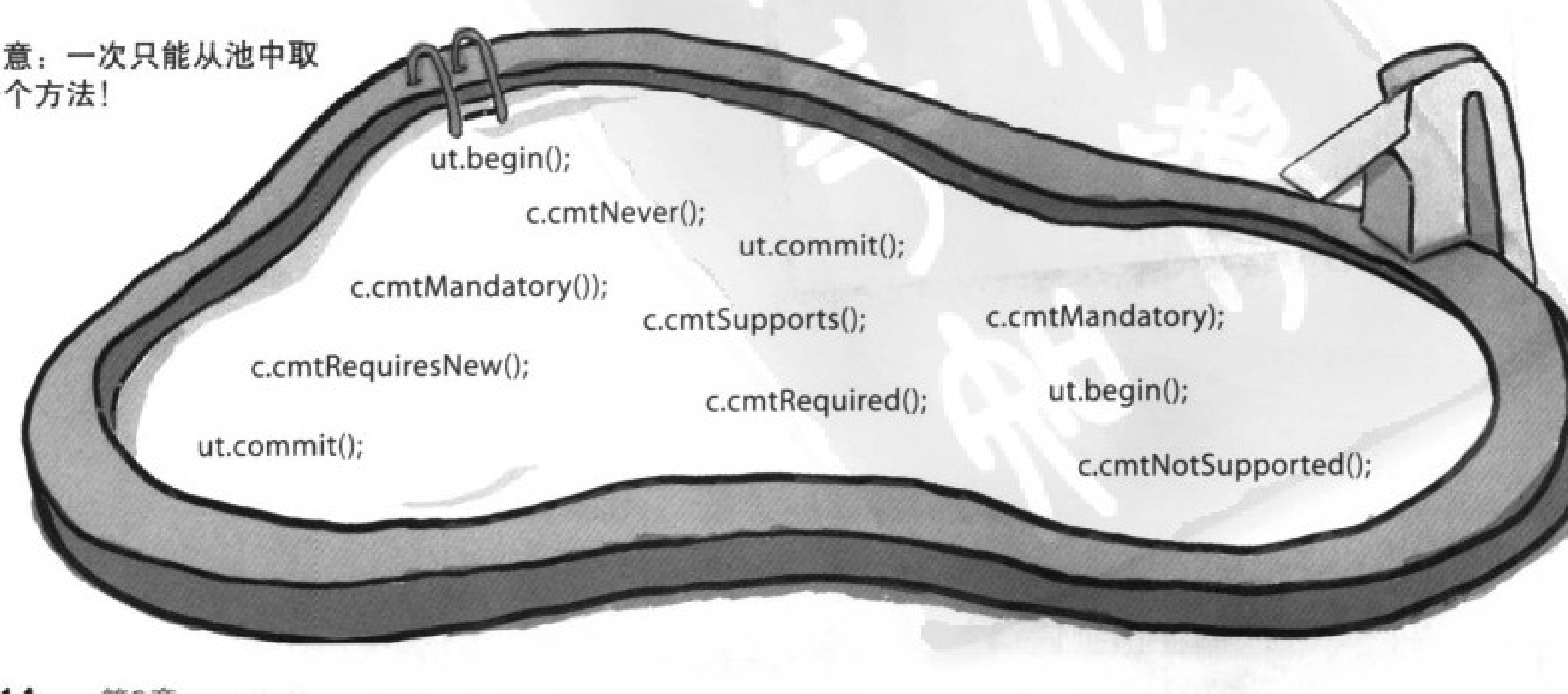
  ____________________

  ____________________

  ____________________

  ____________________
}
```

注意：一次只能从池中取一个方法！



# 池填空答案

(当然，在完成上一页的填空之前，先不要看这个答案。)

```
void bmtMethodWithTransactions () {
    ut.begin();                  tx 1
    c.cmtRequiresNew();          tx 2, 挂起 tx 1
    c.cmtMandatory);             tx 1.（当前一个方法完成时tx 2结束）
    c.cmtRequiresNew();          tx 3, 挂起 tx 1 （这是第二次，也是最后一次事务挂起）
    ut.commit();                 现在没有事务
    c.cmtNotSupported();         还是没有事务
    c.cmtNever();                仍然没有事务
    c.cmtRequired();             tx 4
    ut.begin();                  tx 5 （当前一个方法完成时tx 4结束）
    c.cmtSupports();             tx 5
    c.cmtMandatory());           tx 5
    ut.commit();                 完成
}
```

# COFFEE CRAM
# 模拟测验

---

**1** 关于EJB 2.0中的事务，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. EJB容器必须支持JTA和JTS
- ❑ B. EJB 2.0支持嵌套事务
- ❑ C. `javax.transaction.UserTransaction` API允许设置隔离级别
- ❑ D. bean实例可以并行地运行多个事务
- ❑ E. 消息驱动bean实例必须在“onMessage”方法返回前完成事务

---

**2** 使用BMT划分时，在初始化事务的方法返回之前，以下哪一个必须提交事务？（选出所有正确的答案。）

- ❑ A. 消息驱动bean
- ❑ B. 有状态会话bean
- ❑ C. 无状态会话bean
- ❑ D. 以上都不对

---

**3** 关于EJB 2.0中的容器托管事务，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 在bean的部署描述文件中可以区别重载方法
- ❑ B. bean类中的每个业务方法都必须有一个事务属性
- ❑ C. 如果一个onMessage()方法在提交事务之前先返回了，容器会抛出一个异常
- ❑ D. 使用CMT划分的消息驱动bean不能调用EJBContext.getUserTransaction()方法

**4** 关于在部署描述文件中指定事务属性，以下哪个说法是正确的？（选出所有正确的答案。）

- ❑ A. 对于会话bean，事务属性仅应用于bean组件接口中的方法
- ❑ B. <method-name>标记可以取通配符
- ❑ C. 一个方法在部署描述文件中可以指定多个事务属性
- ❑ D. 不能对实体bean home接口中的方法指定事务属性

**5** 以下哪个事务属性可能导致一个正在进行中的事务挂起？（选出所有正确的答案。）

- ❑ A. NotSupported
- ❑ B. Required
- ❑ C. Supports
- ❑ D. RequiresNew
- ❑ E. Mandatory
- ❑ F. Never

**6** 使用哪个事务属性可能导致抛出一个**`javax.transaction.TransactionRequiredException`**异常？

- ❑ A. NotSupported
- ❑ B. Required
- ❑ C. Supports
- ❑ D. RequiresNew
- ❑ E. Mandatory
- ❑ F. Never

**7** 如果一组 CMT bean方法有以下事务属性：

方法-1 = Supports
方法-2 = Required
方法-3 = NotSupported
方法-4 = RequiresNew

在下面的图中，箭头表示左边的方法调用右边的方法，"tx"号指示一个惟一的事务，对于以上所列的事务属性，下面哪一个图是可能的？（选出所有正确的答案。）

- ❑ A. M1 (Tx 1) -> M2 (No Tx)-> M3 (No Tx) -> M4 (Tx 2)
- ❑ B. M1 (No Tx) —> M2 (Tx 1)
  
  \\—> M3 (Tx 1)
  
  \—> M4 (Tx 2)
- ❑ C. M1 (No Tx) -> M2 (Tx 1) -> M3 (No Tx)
  
  \—> M4 (Tx 2)
- ❑ D. M1 (Tx 1)-> M2 (Tx 1) -> M3 (Tx 1) -> M4 (Tx 2)

**8** 对于使用容器托管事务划分的会话bean，哪些方法在未指定的事务上下文中运行？（选出所有正确的答案。）

- ❑ A. **ejbActivate()**
- ❑ B. **ejbPassivate()**
- ❑ C. 标记为"**NotSupported**"的业务方法
- ❑ D. **ejbRemove()**
- ❑ E. 标记为"RequiresNew"的业务方法

**9** 如果一个会话bean的业务方法调用了**EJBContext.setRollbackOnly()**，下面哪一种事务属性设置可能导致容器抛出**java.lang.IllegalStateException**？（选出所有正确的答案。）

- ❑ A. NotSupported
- ❑ B. Required
- ❑ C. Supports
- ❑ D. RequiresNew
- ❑ E. Mandatory
- ❑ F. Never

**10** 一个使用bean托管事务划分的bean可以成功地调用下面哪个方法？

- ❑ A. **`getUserTransaction()`**
- ❑ B. **`afterBegin()`**
- ❑ C. **`afterCompletion()`**
- ❑ D. **`getRollbackOnly()`**

# COFFEE CRAM 模拟测验答案

(规范：340～341)

**1** 关于EJB 2.0中的事务，哪些说法是正确的？（选出所有正确的答案。）

- [ ] A. EJB容器必须支持JTA和JTS — 仅JTA的UserTransaction
- [ ] B. EJB 2.0支持嵌套事务
- [ ] C. `javax.transaction.UserTransaction` API允许设置隔离级别 — 不行
- [ ] D. bean实例可以并行地运行多个事务
- [x] E. 消息驱动bean实例必须在“onMessage”方法返回前完成事务 — 对！

(规范：340～341)

**2** 使用BMT划分时，在初始化事务的方法返回之前，以下哪一个必须提交事务？（选出所有正确的答案。）

- [x] A. 消息驱动bean
- [ ] B. 有状态会话bean
- [x] C. 无状态会话bean
- [ ] D. 以上都不对

(规范：353～356)

**3** 关于EJB 2.0中的容器托管事务，以下哪些说法是正确的？（选出所有正确的答案。）

- [x] A. 在bean的部署描述文件中可以区别重载方法
- [x] B. bean类中的每个业务方法都必须有一个事务属性
- [ ] C. 如果一个onMessage()方法在提交事务之前先返回了，容器会抛出一个异常 — 不会
- [x] D. 使用CMT划分的消息驱动bean不能调用EJBContext.getUserTransaction()方法

**4** 关于在部署描述文件中指定事务属性，以下哪个说法是正确的？（选出所有正确的答案。） （规范：351～354）

☑ A. 对于会话bean，事务属性仅应用于bean组件接口中的方法

☑ B. <method-name>标记可以取通配符

☐ C. 一个方法在部署描述文件中可以指定多个事务属性

☐ D. 不能对实体bean home接口中的方法指定事务属性 — 必须指定

---

**5** 以下哪个事务属性可能导致一个正在进行中的事务挂起？（选出所有正确的答案。） （规范：357～359）

☑ A. NotSupported

☐ B. Required

☐ C. Supports

☑ D. RequiresNew

☐ E. Mandatory

☐ F. Never

---

**6** 使用哪个事务属性可能导致抛出一个`javax.transaction.TransactionRequiredException`异常？ （规范：357～359）

☐ A. NotSupported

☐ B. Required

☐ C. Supports

☐ D. RequiresNew

☑ E. Mandatory

☐ F. Never

**7** 如果一组 CMT bean方法有以下事务属性： (规范：357～359)

方法-1 = Supports
方法-2 = Required
方法-3 = NotSupported
方法-4 = RequiresNew

在下面的图中，箭头表示左边的方法调用右边的方法，“tx”号指示一个惟一的事务，对于以上所列的事务属性，下面哪一个图是可能的？（选出所有正确的答案。）

- ❑ A. M1 (Tx 1) –> M2 (No Tx)–> M3 (No Tx) –> M4 (Tx 2)
- ❑ B. M1 (No Tx) —> M2 (Tx 1)
  \\\\—> M3 (Tx 1)
  \\—> M4 (Tx 2)
- ☑ C. M1 (No Tx) –> M2 (Tx 1) –> M3 (No Tx)
  \\—> M4 (Tx 2)
- ❑ D. M1 (Tx 1)–> M2 (Tx 1) –> M3 (Tx 1) –> M4 (Tx 2)

**8** 对于使用容器托管事务划分的会话bean，哪些方法在未指定的事务上下文中运行？（选出所有正确的答案。） (规范：363～364，376)

- ☑ A. **ejbActivate()**
- ☑ B. **ejbPassivate()**
- ☑ C. 标记为“**NotSupported**”的业务方法
- ☑ D. **ejbRemove()**
- ❑ E. 标记为“RequiresNew”的业务方法

**9** 如果一个会话bean的业务方法调用了**EJBContext.setRollbackOnly()**，下面哪一种事务属性设置可能导致容器抛出**java.lang.IllegalStateException**？（选出所有正确的答案。） (规范：360～361)

- ☑ A. NotSupported
- ❑ B. Required
- ☑ C. Supports
- ❑ D. RequiresNew
- ❑ E. Mandatory
- ☑ F. Never

**10** 一个使用bean托管事务划分的bean可以成功地调用下面哪个方法？ （规范：361）

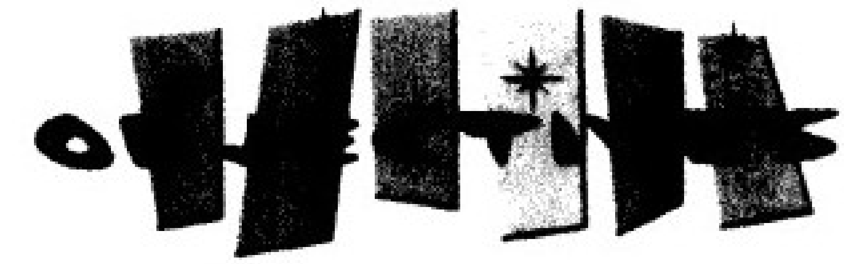
- ☑ A. **`getUserTransaction()`**
- ☐ B. **`afterBegin()`** } 这些是来自
- ☐ C. **`afterCompletion()`** } SessionSynchronization的回调
- ☐ D. **`getRollbackOnly()`** — 这个仅用于CMT bean

# 10 EJB中的异常

## bean要是坏了




**要做最坏的打算。**尽管你竭尽全力，但事情还是有可能会出错。也许很可怕，很悲惨，错得很厉害。你要保护自己，而且可能别人还依赖着你。不能只是因为一个bean抛出一个异常就使得整个程序都陷于崩溃。**应用必须继续下去。**悲剧是无法避免的，但是你可以做好准备。你要知道哪些可以恢复，哪些不能恢复，另外要知道如果出现一个问题该归咎于谁。bean提供者该不该告诉容器还有希望？容器要告诉客户再试一次吗？或者容器干脆删除这个bean，让系统管理员以后再选一个bean……



# 异常

| 标准大纲： | | 具体含义： |
|---|---|---|
| 12.1 | 关于EJB中的异常处理，区别哪些说法或示例是正确的，哪些不正确。 | 必须知道应用异常是客户预期要处理的异常，可以从中恢复，而系统异常则是客户无法恢复的。你要知道5个标准的EJB应用异常，还要知道所有常用的系统异常。你必须准确地了解bean抛出一个应用异常时容器有怎么的表现，当bean抛出一个系统异常时，容器又有什么行为，你还要知道本地客户的异常和远程客户的异常之间有什么区别。 |
| 12.2 | 给定与异常有关的一组职责，明确哪些是bean提供者的职责，哪些是容器的职责。要准备好找出既不是bean提供者也不是容器的职责。 | 你要知道，如果bean抛出一个应用异常，容器不会自动回滚事务，而且根据bean怎么抛出这个异常，容器会按原样把异常交给客户。<br>不过，对于系统异常，容器总是会回滚事务，如果是远程客户，会给它一个RemoteException异常，如果是本地客户，则给它一个EJBException异常。你要知道，如果一个bean希望抛出应用异常会导致事务回滚，那么这个bean必须调用setRollbackOnly()，因为回滚不会自动发生。你要知道容器只会把系统异常记入日志，而不会记录应用异常，另外容器会把产生系统异常的bean杀死（删除）。 |
| 12.3 & 12.4 | 关于实体bean、会话bean和消息驱动bean中的应用异常和系统异常，区别哪些说法或示例是正确的，哪些不正确。<br>给定一个特定的方法条件，明确以下内容：是否会抛出一个异常，所抛出异常的类型，容器的动作，以及客户的视图。 | 如果你想正确地回答这些问题，必须知道前面几章介绍的有关bean的细节内容。例如，你要知道消息驱动bean没有客户视图（即客户接口），所以消息驱动bean不能声明或抛出任何应用异常。你还要知道其他一些内容，例如，如果bean想在其上下文上调用方法，但此时这些操作是不允许的，这个bean就会得到一个IllegalStateException异常。 |

# 哪些地方会出问题？

## ① bean中

- 一个方法中的业务逻辑认识到，由于存在客户预期的一个问题，它无法完成任务。例如，一个银行bean无法完成转账，原因是“源账户”没有钱了！
- bean在运行业务逻辑的时候捕获到一个受查异常。例如，bean无法得到一个JDBC连接，或者JNDI查找向bean抛出一个NamingException异常。这是bean预期的问题，而不是客户预期的。
- bean中的一个方法（或者这个bean调用的一个方法）抛出一个运行时异常（如NullPointerException），但bean没有捕获，而且这个异常也不是客户预期的问题。

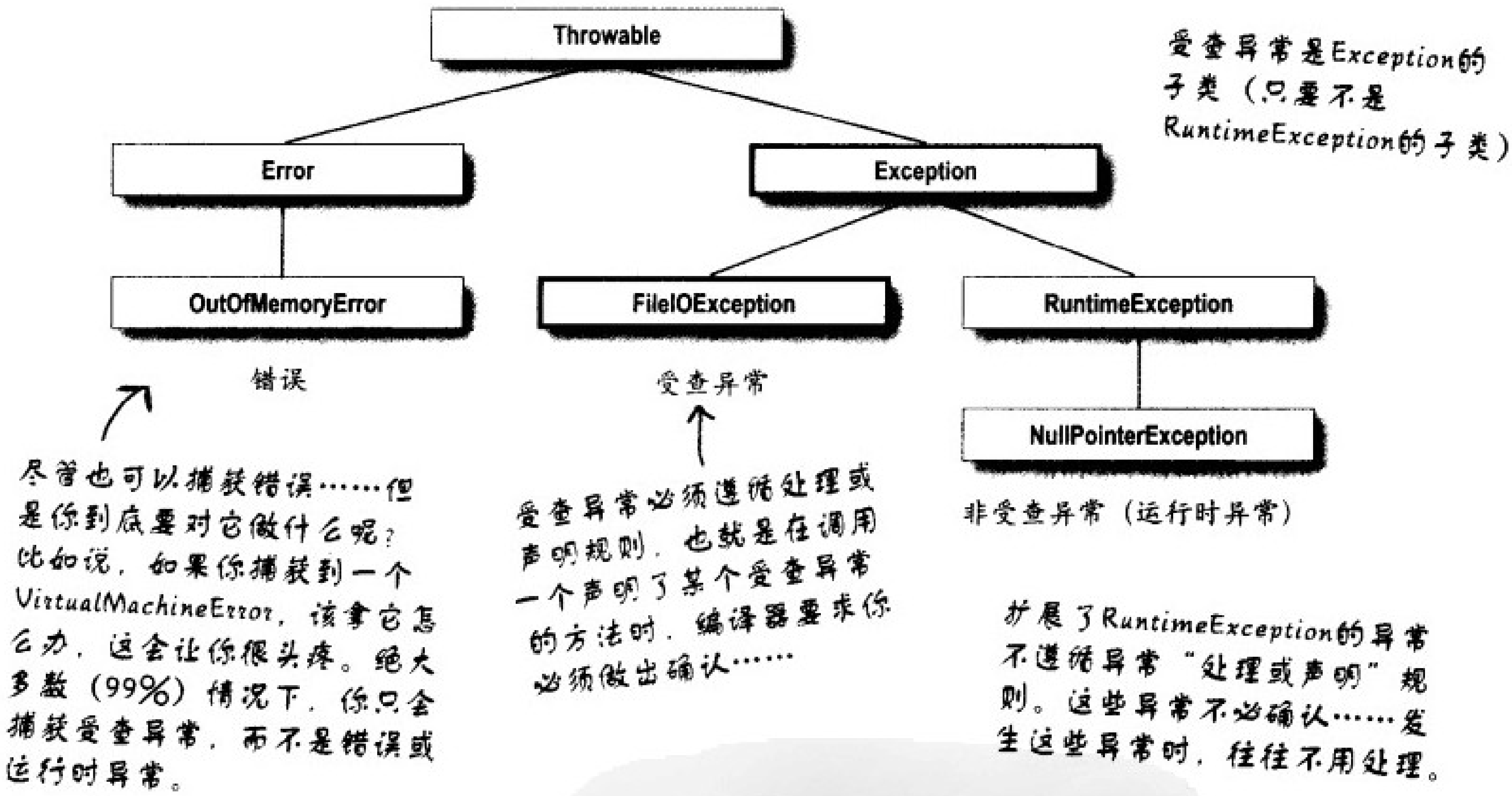
## ② 容器中

- 容器无法完成它负责的操作，如无法在数据库中更新一个实体bean的状态，而这会导致客户预期的一个问题。
- 容器捕获到bean抛出的一个受查异常，这也是客户预期的异常。
- 容器捕获到bean抛出的一个运行时异常，或者与容器交互的其他对象抛出的运行时异常。这不是客户预期的问题。

## ③ RMI子系统中，或者客户和容器之间的通信路径上的其他部分

- EJB对象向客户传递结果（即客户对bean的一个业务方法调用的结果）之前抛出一个运行时异常。
- RMI子系统无法与客户通信。
- 试图发出一个方法调用或者接收一个返回值时，客户桩抛出一个运行时异常，或者RemoteException。

# 记住，Java异常可以是受查异常，也可以是非受查异常




我们知道，这些你都已经记住了，不过正因如此我们才有共同语言，我们强调的就是Java有受查异常也有非受查异常。受查就是编译器会检查。如果你调用了一个方法，这个方法声明了一个抛出受查异常的throws子句，就必须向编译器保证你很清楚这个潜在的问题，而且已经准备好冒这个险。这就是处理或声明（handle or declare）规则。要么把这个有风险的方法调用包装在一个try/catch中，要么只是声明再抛出这个异常。

要记住，声明一个异常意味着躲开这个异常，让调用栈中别的方法来处理它。声明异常就如同说“我不想捕获这个异常，因为我觉得别人可以更好地处理它（或者应该由别人来处理）”。

如果你处理了一个异常并退出catch块。那么异常是否真的发生应该没有什么区别。如果发生了异常和未发生异常的结果不一样，这说明你没有正确地处理这个异常，可能应该避开它（把它交给别人来处理）。

运行时异常是非受查异常（只在运行时有影响，而不是编译时）。可以在代码中用任何方式声明、捕获和抛出这种异常，不过编译器并不关心。但是，如果你的代码导致一个运行时异常，你所运行的调用栈/执行线程就会死。如果这是程序中最后一个非守护线程，整个程序就会关闭。

# 都与预期有关……

EJB（或者更一般地，Java）中的异常处理都以预期为中心。你希望一切都能正确工作，希望不要得到一个运行时异常。但是你有冒险行为，所以预期到可能有些地方会出问题。关键是要知道哪些地方会出问题，哪些是可以恢复的。

在大多数情况下，出问题的地方是很明显的。你做了一个JNDI查找，但是没有这样的对象。你想连接一个命名服务，但是连不上。你想连接数据库，但是无法连上。你想创建一个新的实体bean，但是插入操作失败了，因为存在重复的主键。而且，对于上述种种情况，都能编写一个catch块，其中知道如何处理遇到的特定问题。连不上命名服务吗？也许你可以连接另一个命名服务试试看。无法连接数据库吗？你现在可以先使用一个本地缓存，直到能重新建立连接为止。无法完成插入？再使用另外一个主键试试。

为一个有风险的方法调用提供catch块时，就是在说你预期到可能会有某个特定的问题出现。几乎所有提供了相应catch块的异常都是能预期的异常。所以对于上面的示例，你可以预期到这分别是NamingException、FinderException或CreateException异常。

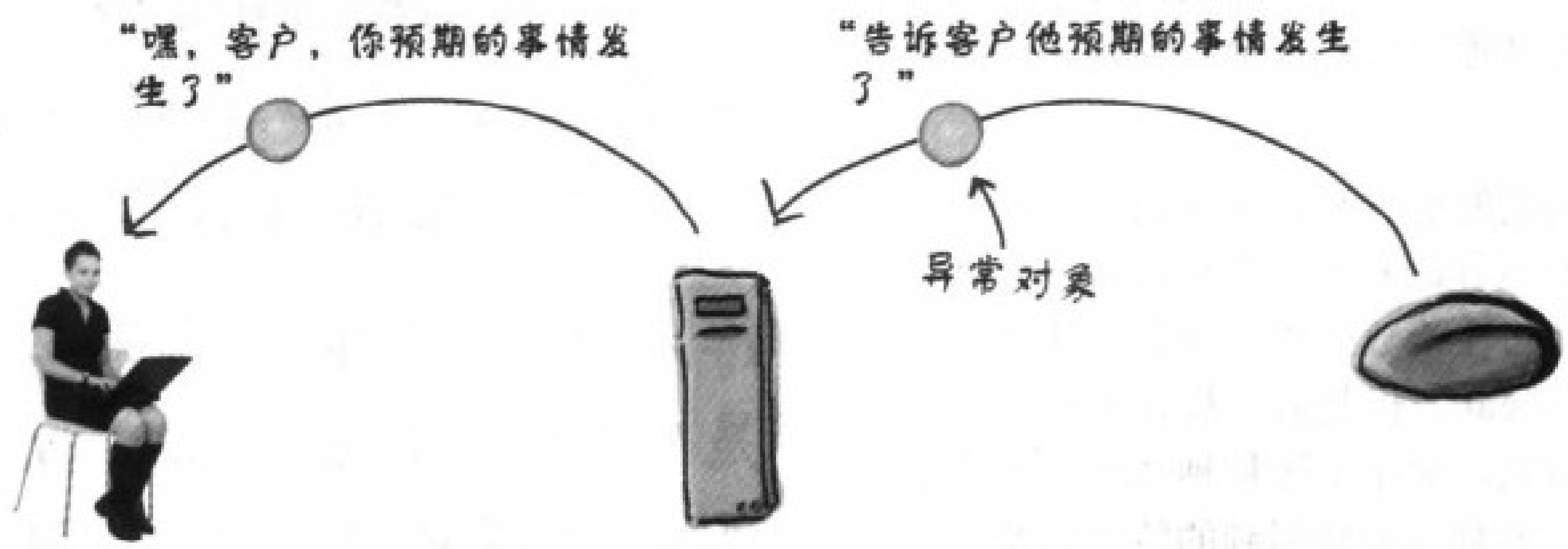
**把一个有风险的方法调用包装在一个try/catch块中，这就告诉编译器（和你的设计）你预期某些事情可能会出错。**

**你希望一切都能正确工作，但是不能保证这一点，所以针对你认为可能出错的问题必须有一组catch块……**

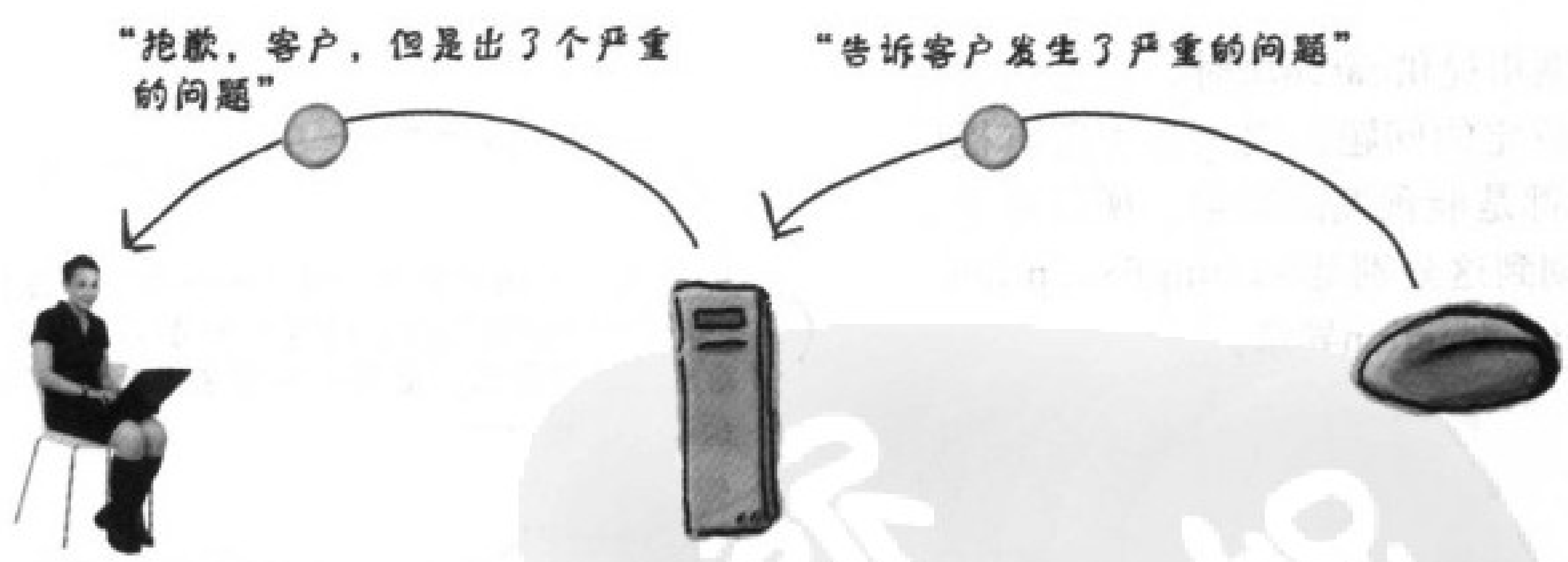


# 异常之路

## ① bean抛出客户预期的异常



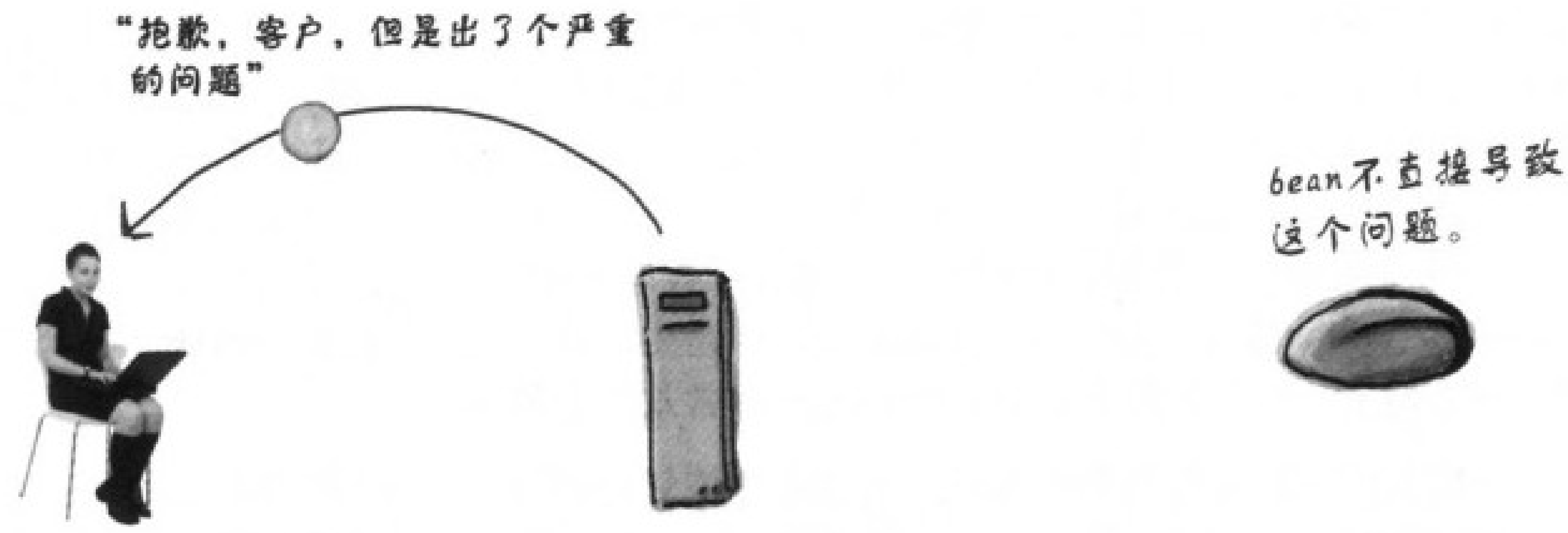

## ② bean抛出客户没有预期的异常




## ③ 容器抛出客户预期的异常



④ 容器抛出客户没有预期的异常




⑤ 桩抛出客户预期的异常




⑥ 桩抛出客户没有预期的异常



# EJB中，异常有两类：应用异常和系统异常

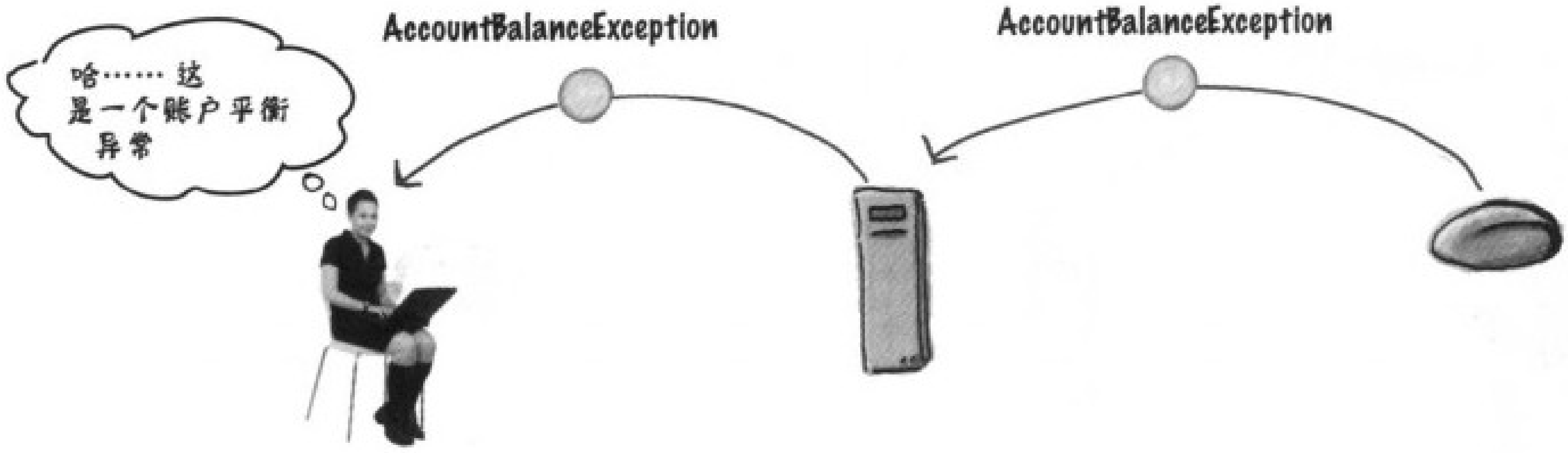
应用异常是客户预期的异常，这意味着客户必须采用某种方式处理这些异常。这也说明应用异常在展现给客户的接口中声明，而且客户调用这个方法时可以使用一个try/catch块。由于客户可以捕获到预期的异常，所以客户可以从中恢复并继续工作。应用异常包括CreateException（也许客户提供的参数不合法），或者AccountBalanceException（可能客户下一次调用会访问另一个不同的账户），或者ObjectNotFoundException（数据库中不再有这个实体），等等。应用异常包括bean或容器抛出的所有受查异常，但java.rmi.RemoteException除外。尽管RemoteException 是受查异常，而且由客户捕获，但对客户而言，服务器上是什么导致了RemoteException异常却是未预期的。

系统异常对应客户没有预期而且/或者无法从中恢复的问题。发生在服务器上的几乎所有运行时异常都是系统异常（从bean、容器或服务器上任何其他对象产生的异常），或者甚至客户本地桩对象产生的异常也是系统异常。系统异常是客户确实无法恢复的问题，可能是由于它们不可恢复（如数据库中的一个失败，或者NullPointerException），也可能是因为客户没有足够的信息来决定该做什么（如对于一个RemoteException，尽管这是客户捕获的一个受查异常，但是只是告诉客户服务器上发生了某个未预期的不好的事情）。

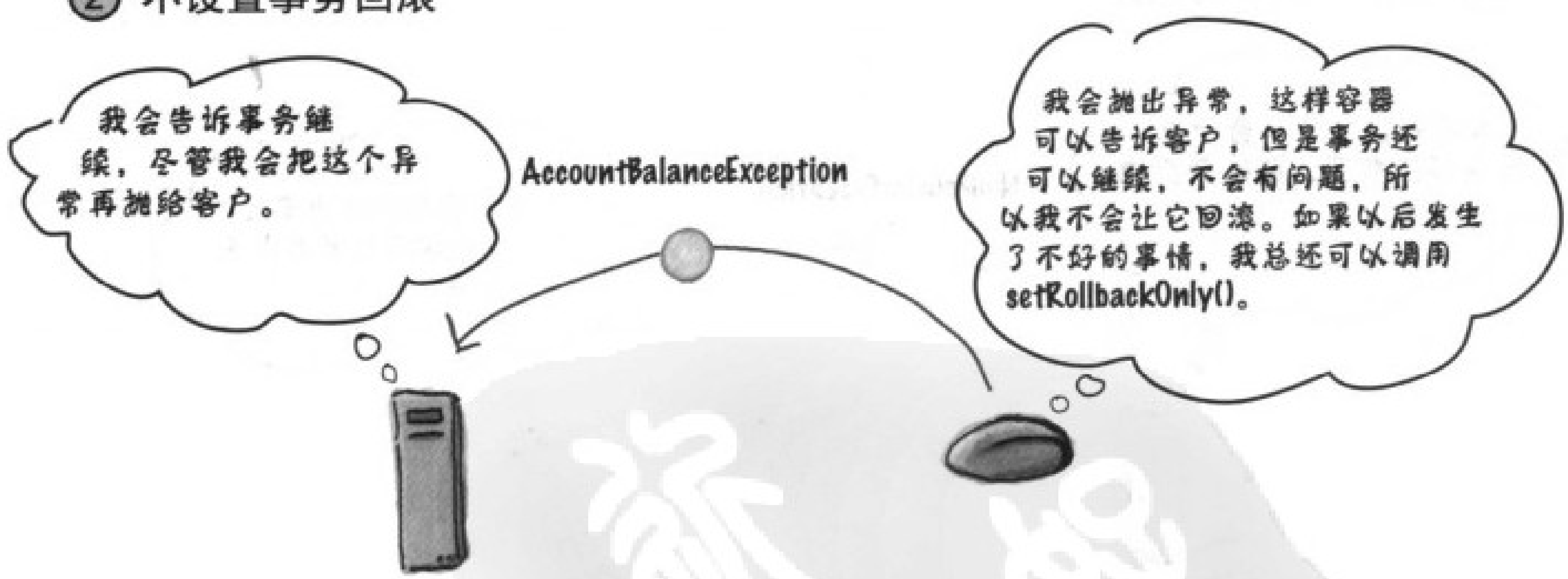


# 对于一个应用异常，容器会……

① 照原来抛出的样子原封不动地把它发回给客户




② 不设置事务回滚

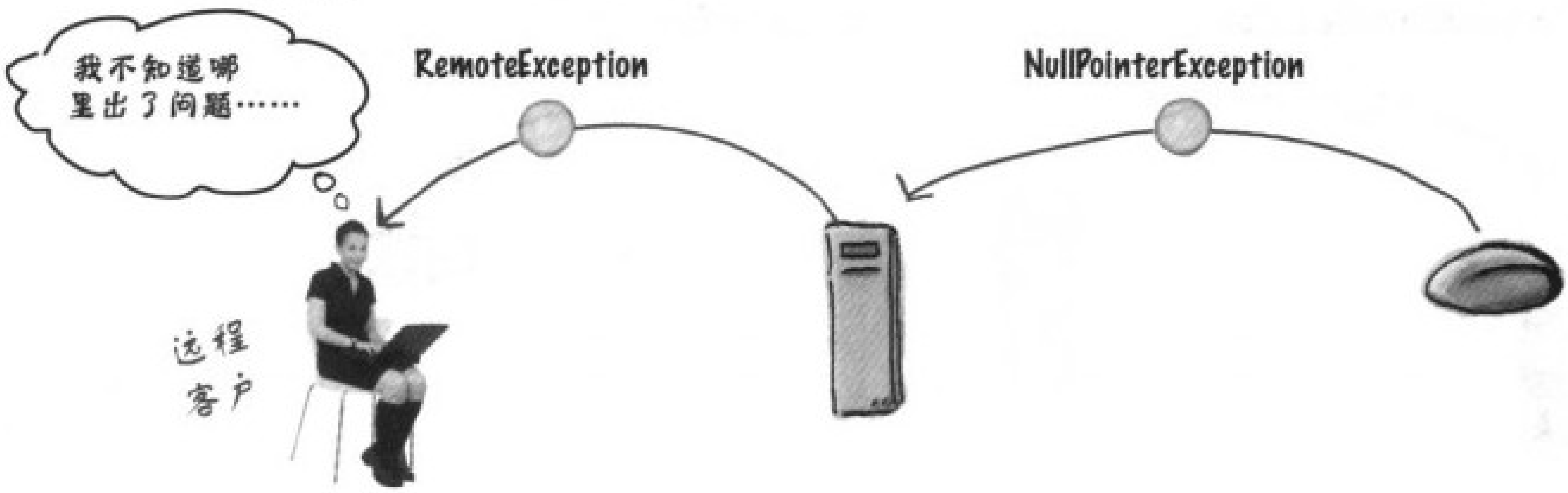



③ 让bean继续活下去

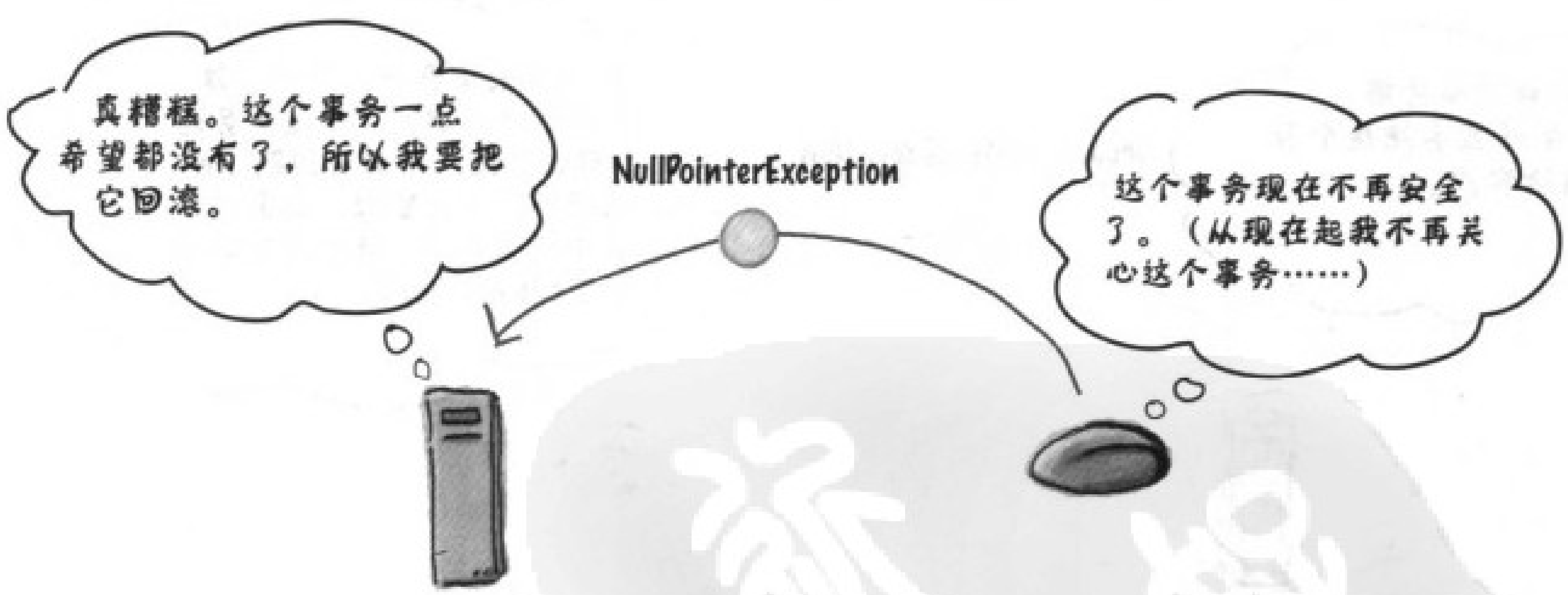


# 对于一个系统异常，容器会……

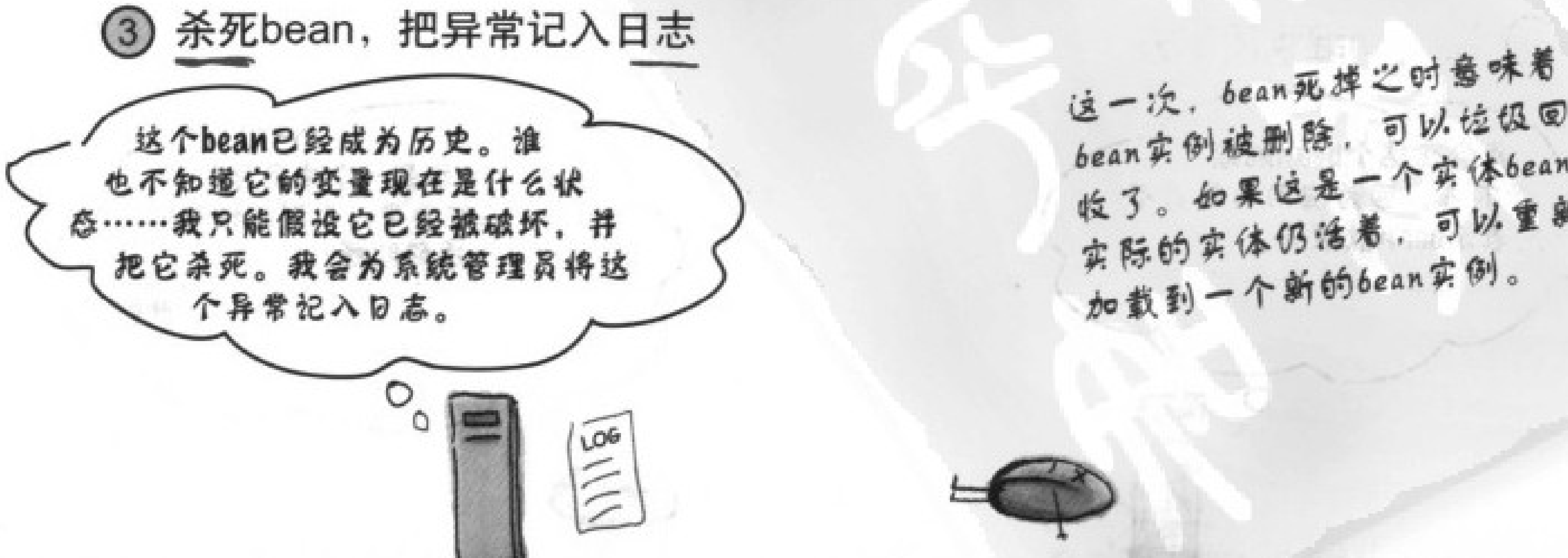
① 把它作为一个RemoteException发回给远程客户，或者作为一个EJBException发回给本地客户

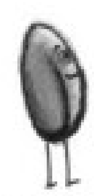
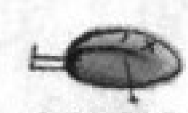
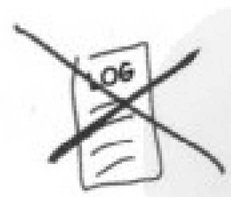
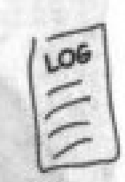



② 总是导致事务回滚




③ 杀死bean，把异常记入日志



| | 应用异常 | 系统异常 |
|---|---|---|
| 客户恢复 |  客户可以尝试恢复 |  客户无法恢复 |
| 事务状态 | A<br>事务继续进行（除非bean调用了setRollbackOnly()） | <br>自动回滚 |
| bean实例 | <br>实例活着 | <br>实例死掉 |
| 日志 | <br>不记日志 | <br>异常记入日志 |
| 示例 | CreateException<br>RemoveException<br>FinderException<br>ObjectNotFoundException<br>DuplicateKeyException<br><br>AccountBalanceException<br>BadQueryArgsException<br>BookMappingException<br>Exception | RemoteException（远程客户）<br>EJBException（本地客户）<br>IllegalStateException<br>TransactionRequiredException<br>NoSuchObjectException<br>ArrayIndexOutOfBoundsException<br>NullPointerException<br>RuntimeException |
| 编译器检查和其他规则 | 所有应用异常都必须是受查异常，而且不能扩展RemoteException。 | 惟一一个属于系统异常的受查异常是RemoteException。所有其他系统异常都是RuntimeException（或它的一个子类）。 |




## BRAIN POWER

如果你知道你想向客户抛出一个应用异常，但是不希望事务提交，你能做些什么？

先考虑一下。

后面几页我们将讨论这种情况。

# 警告！RemoteException是受查异常，但不是预期的！

通常我们在考虑预期和未预期的异常时，往往把它们对应为受查异常和非受查异常。FileIOException是预期的异常。NullPointerException则不是。Exception的任何直接子类都是预期的异常。RuntimeException的子类则不是。受查异常必须得到处理，非受查异常往往不会处理。

但是有关异常和预期的问题，有一个很突出的例外，这就是java.rmi.RemoteException。

当然，RemoteException是一个受查异常，所以客户必须用一个try/catch块来处理它，从而对RemoteException做出确认。不过，不同于客户在bean接口中看到的其他受查异常，RemoteException仍被认为是未预期的。应该说并非未预期到这个异常本身……而是这个异常对应的具体情况客户无法预见到。

从客户的观点来看，可以把RemoteException看作是应用远程部分（服务器）上的一种运行时异常，这是因为通常RemoteException的含义就是如此。服务器上的一个NullPointerException表示对客户的一个RemoteException。服务器上的DivideByZeroException对客户来说也是一个RemoteException，服务器上的ClassCastException对客户来说同样是一个RemoteException。所有这些都是服务器上的非受查异常，不过它们都作为受查的RemoteException传回给客户。换句话说，客户必须预期到服务器可能抛出一些未预期的异常。

这是不是意味着客户得到的每个RemoteException最初都是由于bean得到一个运行时异常而触发的呢？不是这样的，例如，一个bean可以把捕获一个受查异常作为它的业务逻辑的一部分，后来发现这个异常无法恢复。此时，这个bean会把原来的受查异常（在bean看来是受查异常）转变成一个非受查异常（系统异常），也就是把原来的受查异常包装起来并作为一个EJBException重新抛出，最终这就会在远程客户上以RemoteException的面貌出现。

实际上，客户即使没有服务器，也可能得到一个RemoteException。例如，如果桩无法到达服务器，它就可能抛出一个RemoteException。关键是要把应用异常看作是客户必须确认而且可以恢复的受查异常，而所有其他的异常都是系统异常，其中也包括RemoteException（以及它的子类）。

**应用异常总是编译器检查的异常，但RemoteException除外。**

**可以把RemoteException看作是“一个服务器上的运行时异常”，尽管RemoteException对客户来说是一个受查异常。**

**换句话说，远程客户必须预期到服务器可能抛出一些未预期的异常。**

## 远程实体bean home接口声明了应用异常和一个系统异常(RemoteException)

```
package headfirst;

import javax.ejb.*;
import java.rmi.RemoteException;
import java.util.Collection;

public interface CustomerHome extends EJBHome {

   public Customer create(String last, String first) throws CreateException, RemoteException;

   public Customer findByPrimaryKey(String key) throws FinderException, RemoteException;

   public Collection findByCity(String city) throws FinderException, RemoteException;
}
```

应用异常（指向 CreateException）

系统异常（指向 RemoteException）

客户预期而且可能可以恢复的异常（这是一个应用异常）

服务器上发生的未预期的问题，所以客户无法恢复。

## 本地实体bean home接口只能声明应用异常

```
package headfirst;

import javax.ejb.*;
import java.util.Collection;

public interface CustomerHomeLocal extends EJBLocalHome {

   public Customer create(String last, String first) throws CreateException;

   public Customer findByPrimaryKey(String key) throws FinderException;

   public Collection findByCity(String city) throws FinderException;
}
```

应用异常（指向 CreateException）

应用异常（指向 FinderException）

# RemoteException交给远程客户 EJBException交给本地客户

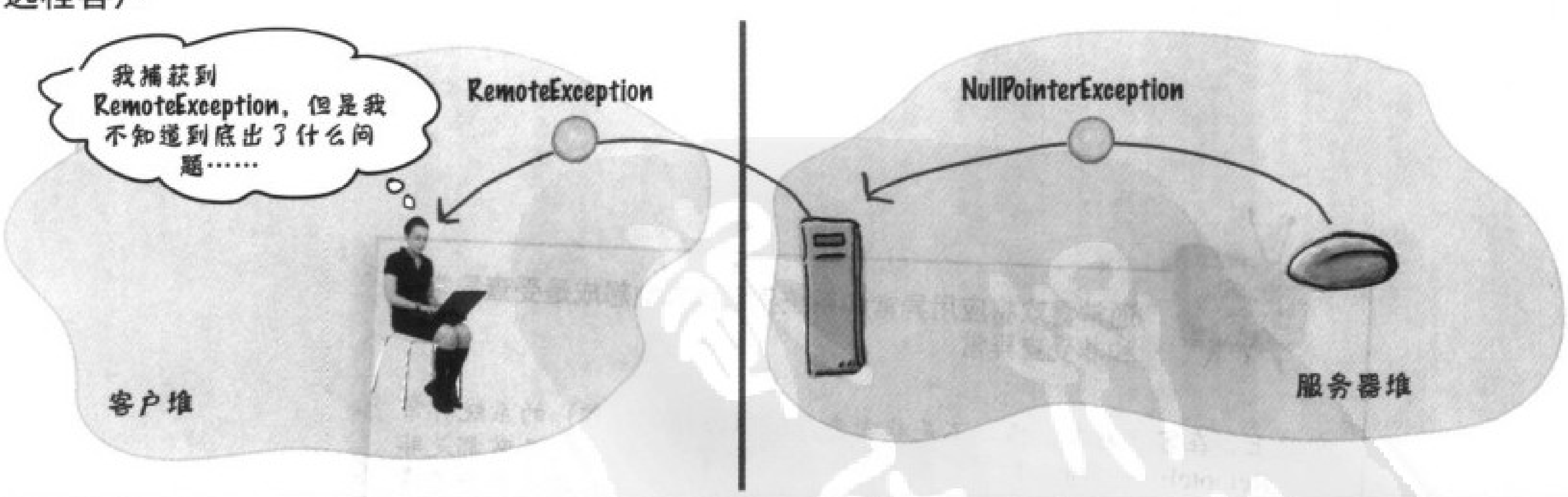
如果服务器上发生了未预期的事情，容器会抛出一个RemoteException或一个EJBException来通知客户。

RemoteException只交给远程客户，而且尽管它是一个受查异常，但是会告诉客户发生了未预期的事情，可能是客户无法恢复的问题。

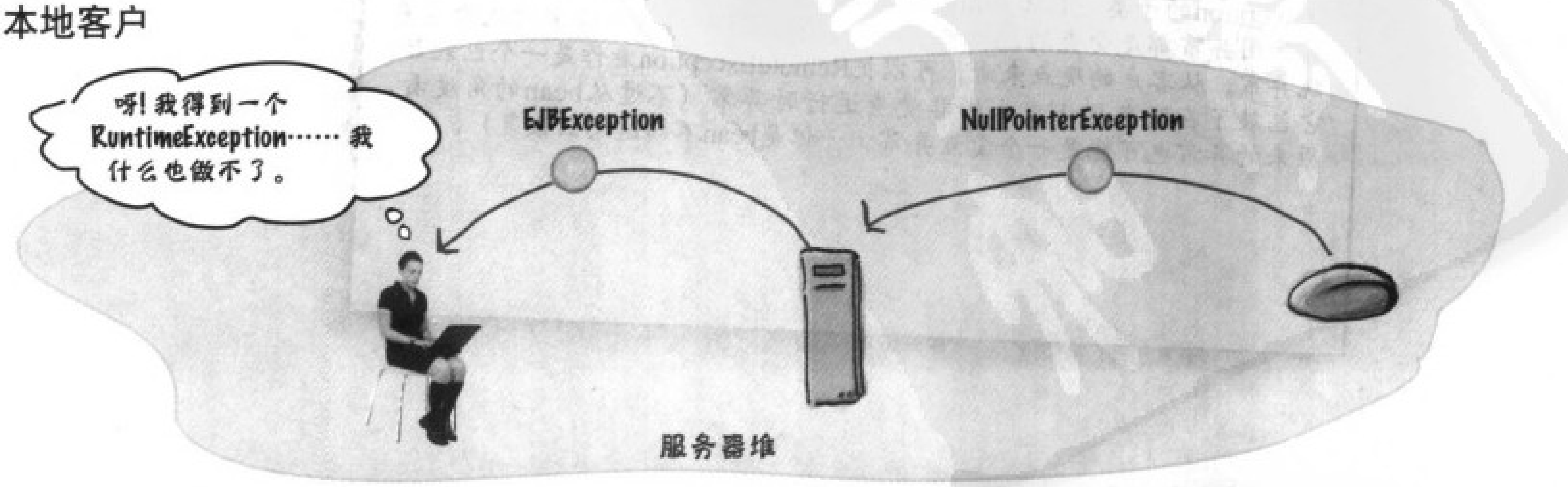
EJBException只交给本地客户，而且它是非受查的。换句话说，EJBException是RuntimeException的一个子类。对于客户来说，得到一个EJBException与得到其他异常（如ArrayIndexOutOfBoundsException）并没有太大的区别。这说明未预期的地方出了问题，而且对此你没有办法恢复。惟一的区别是，客户确实必须捕获RemoteException，但是一旦捕获到这个异常，客户并不能说出到底发生了什么（至少不知道怎么恢复）。

**当服务器发生了未预期的事情时，本地客户会得到一个RuntimeException（EJBException），而远程客户会得到一个编译器检查的RemoteException。**

远程客户




本地客户



## there are no Dumb Questions

**问：** 在图中，你说客户得到一个RemoteException时想的是“我不知道发生了什么”。但是我在shell终端收到一个RemoteException的时候，它总是会给我一些消息告诉我发生了什么事情，比如“服务器上出现一个NullPointerException”或者诸如此类的消息……

**答：** 不过在运行时这对你有什么帮助吗？当然，在开发和测试阶段这是很有帮助的，但是应用一旦真正运行起来，你的客户代码可能不会解析栈轨迹。我们说客户不知道发生了什么，意思是说客户无法捕获到特定的东西，也就是无法从中恢复。

这与CreateException之类的异常不同，对于这些异常，客户可以捕获，而且可以从catch块再尝试创建操作，可能会提供一个不同的参数。

**通常喜欢将应用异常和系统异常简单地想成是受查异常和非受查异常。**

实际上，在客户看来，惟一不是非受查异常（运行时异常）的系统异常就是RemoteException。所以，对于本地客户而言，所有系统异常都是非受查异常（RuntimeException的子类），而所有应用异常都是受查异常（Exception的子类，但RuntimeException除外）。对于远程客户而言，所有应用异常都是受查的，但也有一个例外，有一个受查异常被认为是系统异常。从客户的观点来看，可以把RemoteException看作是一个包装器，它包装了在服务器上发生的非受查运行时异常（不过从bean的角度看，原来的异常也可能是一个受查异常……但是bean不知道怎么恢复）。

# bean提供者的职责

## ❶ 如果你的业务逻辑捕获（或创建）一个应用异常，会作为应用异常把它抛给容器。

在bean代码中，如果你捕获到其他代码抛出的一个应用异常（或者你自己创建这个异常，作为业务逻辑的一部分），则把它原样抛给容器。例如，如果你的bean代码在查找另一个bean的时候（换句话说，在另一个bean的home接口上调用一个查找方法时）得到一个FinderException异常，就会原封不动地把这个FinderException异常交给容器。这说明要么声明这个异常（躲开它，让它沿着栈回传，交给容器）：

```
public void someBusinessMethod() throws FinderException {
    CustomerHome home = null;
    try {
        InitialContext ic = new InitialContext();
        Object o = ic.lookup( "Customers" );
        home = (CustomerHome) PortableRemoteObject.narrow(o, CustomerHome.class);
    } catch(NamingException ne) {
          // 处理NamingException
    }
    try {
        Customer cust = home.findByPrimaryKey( "42" );
          // 更多内容
    } catch(RemoteException re) {
         // 处理RemoteException
    }
}
```

如果想抛出这个异常，必须声明它（这是常规的Java法则）

这里捕获了一个NamingException和RemoteException，但是没有捕获FinderException，所以FinderException会自动传播回容器。

要么捕获这个异常，并原样抛出：

```
public void someOtherBusinessMethod() throws FinderException {
    CustomerHome home = null;
    try {
        InitialContext ic = new InitialContext();
        Object o = ic.lookup( "Customers" );
        home = (CustomerHome) PortableRemoteObject.narrow(o, CustomerHome.class);
    } catch(NamingException ne) {
          // 处理命名异常
    }
    try {
        Customer cust = home.findByPrimaryKey( "42" );
          // 更多内容
    } catch(RemoteException re) {
         // 处理这个异常
    } catch(FinderException fe) {
         if (! recoverable()) {
            throw fe;
         } else {  //其他内容}
    }
}
```

当然，还是必须声明

捕获到这个异常，但是如果无法恢复，会原样把它重新抛出

# bean提供者的职责

**② 如果你捕获到一个应用异常，而且发现无法继续这个事务，就要在向容器抛出这个异常之前先调用setRollbackOnly()。**

要记住，容器不会因为有一个应用异常就回滚事务。如果有一个应用异常，容器会假定一切还可以恢复，这个事务还能继续。除非你发现在业务逻辑中继续提交不是一个好想法（一般来说都是这样），此时你要怎么做呢？应当使用EJBContext（对于CMT bean）或UserTransaction（对于BMT bean）上的setRollbackOnly()要求容器回滚事务。

```
public void someOtherBusinessMethod() throws FinderException {
    CustomerHome home = null;
    try {
        InitialContext ic = new InitialContext();
        Object o = ic.lookup( "Customers" );
        home = (CustomerHome) PortableRemoteObject.narrow(o, CustomerHome.class);
    } catch(NamingException ne) {
          // 处理命名异常
    }
    try {
        Customer cust = home.findByPrimaryKey( "42" );
          // more stuff
    } catch(RemoteException re) {
        // 处理这个异常
    } catch(FinderException fe) {
         if (! recoverable()) {
            context.setRollbackOnly();
            throw fe;
         } else { // 其他内容}
    }
}
```

捕获到这个异常，但是我们知道无法恢复，而且不能让事务提交！！

# bean提供者的职责

③ **如果你的业务逻辑捕获到一个客户未预期的异常（也就是说，没有在客户视图中声明的异常），要把它包装起来，并作为一个EJBException重新抛出。**

客户应该对你的内部行为知道多少呢？应该知道一点或者根本不知道，对不对？例如，你在使用JDBC来完成工作，这是不是客户业务的一部分？你会把捕获一个SQLException作为业务逻辑的一部分吗？应该只抛出一个EJBException，这是在以程序员的方式告诉容器“我失控了”。




**下面的做法是合法的，但是不好：**

```
public interface BadService extend EJBLocalObject {
   public void someMethod() throws SQLException;
}
```

你真的认为客户应该知道你在使用SQL吗？

```
public class BadServiceBean implements SessionBean {
   public void someMethod() throws SQLException {
     // 可能导致SQLException的工作
   }
  // 其他bean方法
}
```

---

**应该这样做：**

```
public interface GoodService extend EJBLocalObject {
   public void someMethod();
}
```

```
public class GoodServiceBean implements SessionBean {

public void someMethod() {
   try {
      // 做工作
   } catch (SQLException se) {
        // 如果无法恢复
      throw new EJBException(se);
   }
}
```

EJBException是一个运行时异常，所以不用声明。

包装并作为一个EJBException抛出。

# bean提供者的职责

## ④ 如果你的业务逻辑遇到一个运行时异常，把它传播到容器。不要去捕获它！

千万不要去捕获运行时异常。例如，不要用一个try/catch来捕获Exception。这意味着你要捕获所有的异常，最糟糕的是，你可能会“吞掉”运行时异常，而不是把它们传回给容器。如果你确实要捕获一个运行时异常，但是随后发现根本无法恢复，可以把它原样抛出。但是无论怎样，都不要像下面这样做：

```
public void dumbMethod() {
    try {
      //
    } catch (Exception ex) { }
}
```

这个想法确实很傻。不要捕获任何Exception（当然更不要捕获所有Throwable!），只捕获你的业务逻辑需要的异常，而让其他运行时异常沿着调用栈传回容器。

## ⑤ 如果你的业务逻辑生成一个应用异常，必须在客户接口和bean类中都声明这个异常。

要记住，只是在一个接口中声明了一个异常，这并不意味着你要在方法的实现中声明这个异常。不论是否以常规的Java方式来实现方法（也就是你的类是否声明了“extends ThisInterface”）都是如此。但是，如果你有可能抛出一个应用异常，可能是因为代码中有一个try/catch，也可能是因为你自己的业务逻辑可以创建一个异常，那就必须在bean类中声明这个异常。

```
public void withdraw(double d) throws AccountBalanceException {
    if ((balance - d) < 0) {
       throw new AccountBalanceException(overdrawnMsg);
    } else {
       balance -= d;
    }
  }
}

public abstract void withdraw(double d) throws AccountBalanceException;
```

如果可能抛出这个异常，就必须声明。

# bean提供者的职责

**❽ 如果创建了你自己的应用异常，这些异常必须扩展（直接或间接）Exception，但不能扩展RuntimeException或RemoteException。**

尽管建议你使用或扩展标准EJB异常（稍后将介绍这些异常），但是如果你的设计要求建立自己的定制异常，则必须把这些异常建立为受查异常。这说明，它们必须扩展java.lang.Exception（或Exception的某个子类），而不能扩展java.lang.RuntimeException。还有一个限制，应用异常不能扩展java.rmi.RemoteException（无论是直接还是间接都不允许）。要记住，应用异常是客户视图中声明的任何受查异常，但RemoteException除外。

```
public void withdraw(double d) throws AccountBalanceException {
    if ((balance - d) < 0) {
       throw new AccountBalanceException(overdrawnMsg);
    } else {balance -= d;}
  }
}
class AccountBalanceException extends Exception {
    AccountBalanceException(String s) {
       super(s);
    }
}
```

## there are no Dumb Questions

**问：** **我发现一个问题……SessionBean和EntityBean接口中声明的容器回调没有声明受查异常，对不对？那么这是不是意味着不能从这些回调方法（比如ejbActivate()）抛出应用异常呢？**

**答：** 你说得对！对于不是客户视图一部分的容器回调，只能从这些方法中抛出非受查异常。这只是普通的Java法则……如果一个方法没有声明一个受查异常，就不能从这个方法抛出这个异常，而且由于这些接口没有声明你可能使用的任何受查异常，所以遇到的只能是运行时异常。从SessionBean、EntityBean或MessageDrivenBean中的容器回调只能抛出一个异常，这就是EJBException。

**问：** **你说“接口没有声明你可能使用的任何受查异常”，这是什么意思？还可能声明不能使用的受查异常吗？请举个例子。**

**答：** 如果你去看J2EE API（你尽管去看，我们会等你，一直等你看完……），你会看到所有方法都声明了一个RemoteException！对，这是一个很庞大的受查异常（它们也声明了EJBException，但这不是必需的，因为EJBException是RuntimeException的一个子类）。这是不是说，你可以自己抛出一个RemoteException呢？比如，如果你的bean作为另一个bean的客户时捕获到一个RemoteException，你就把它抛出，这样行不行？不行，绝对不行！

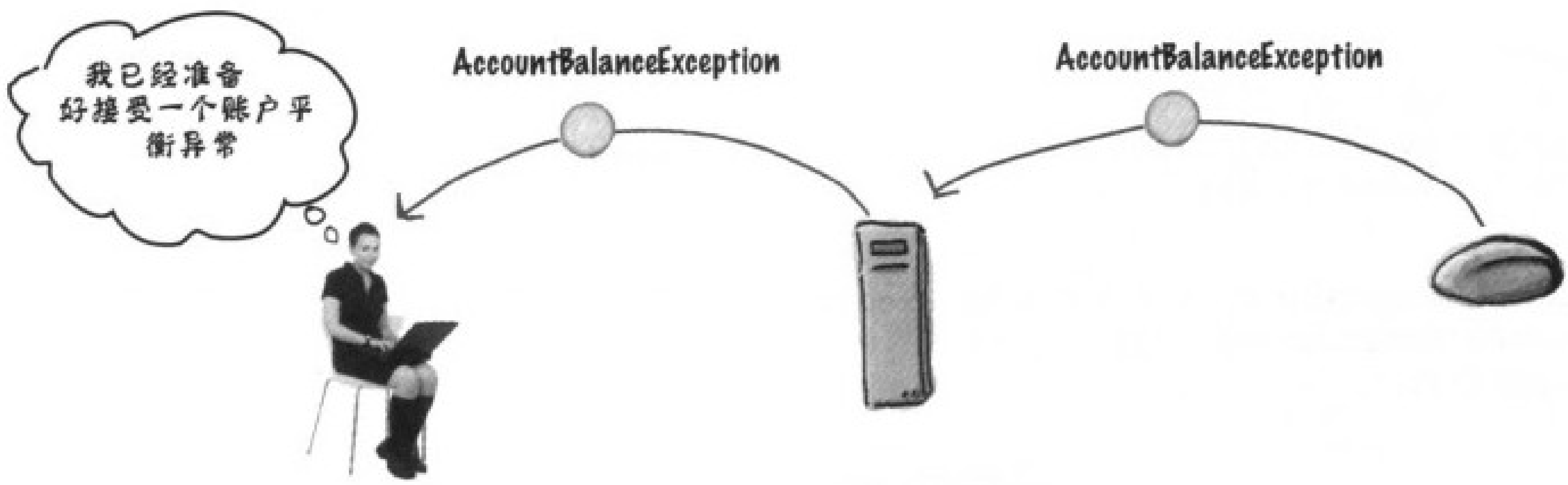
在采用流驱动容器的年代，EJB 1.0规范允许从bean抛出RemoteException。那个时代已经过去了，这在EJB 2.0中是不允许的。所以除非你背负着维护既有bean的任务，否则看到API文档的时候最好转移注意力，假装根本没有看到RemoteException……

# 容器的职责

**① 如果bean抛出一个应用异常，按它抛出的样子原样发回给容器，而且不回滚事务。**

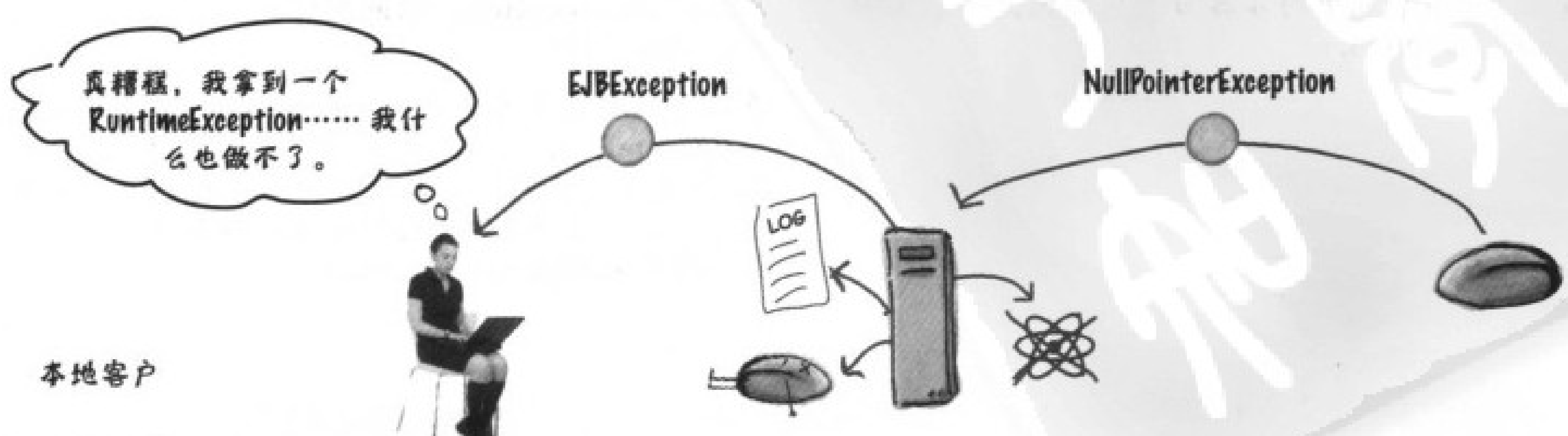
如果bean抛出一个CreateException，把这个异常发送给客户。如果bean抛出一个FinderException，把这个异常发送给客户。如果bean抛出一个AccountBalance-Exception, 还是把这个异常发送给客户。

异常究竟是java.ejb包中的某个标准EJB异常，还是bean提供者定义的一个异常，这并没有什么区别。




**② 如果bean抛出一个系统异常（包括EJBException或任何运行时异常）**

- 如果客户是远程客户，则抛出一个RemoteException
- 如果客户是本地客户，则抛出一个EJBException
- 把异常记入日志
- 回滚事务
- 删除bean实例（就好像丢掉一个吐司一样）



## BRAIN POWER

先考虑一下客户。如果客户得到一个RemoteException，他能肯定地知道业务方法是否完成了吗？

客户能肯定地知道事务是否回滚吗？

客户有没有办法知道呢？
如果客户是另一个bean呢？

## Relax

**考试中不会考处理事务失败的非J2EE客户策略。**

不过，作为一个开发人员，你要考虑在客户端可能发生什么事情，特别是你不知道一个事务是否会成功的情况下，更要仔细考虑。如果你的客户是一个bean，或者是J2EE的另一部分，那很好。J2EE客户通常可以找出事务的状态。但是如果是一个非J2EE客户，就需要另外的某种机制，或者至少要有办法保证操作幂等性，也就是事务已经成功之后（但是客户还不知道），如果客户想再做同样的事务，第二次（或后面的）尝试也不会带来问题。如果成功之后再尝试，这个操作不会带来问题，则意味着你的操作是“幂等的”（idempotent）。虽然这个词读起来不太好听，但对于应用的完整性确实很重要。

## Sharpen your pencil

给定以下可能的选择，作为一个bean提供者，请针对下面的各种场景选出应该怎么做。假设这些情况都在一个会话bean的业务方法中发生。

选择（一个选择可以使用多次）

**A. 抛出一个EJBException**

**B. 抛出一个RemoteException**

**C. 调用setRollbackOnly()**

**D. 允许异常传播（也就是说，躲开它）**

场景

在ejbActivate方法中捕获到一个受查异常。这个方法不在事务中。

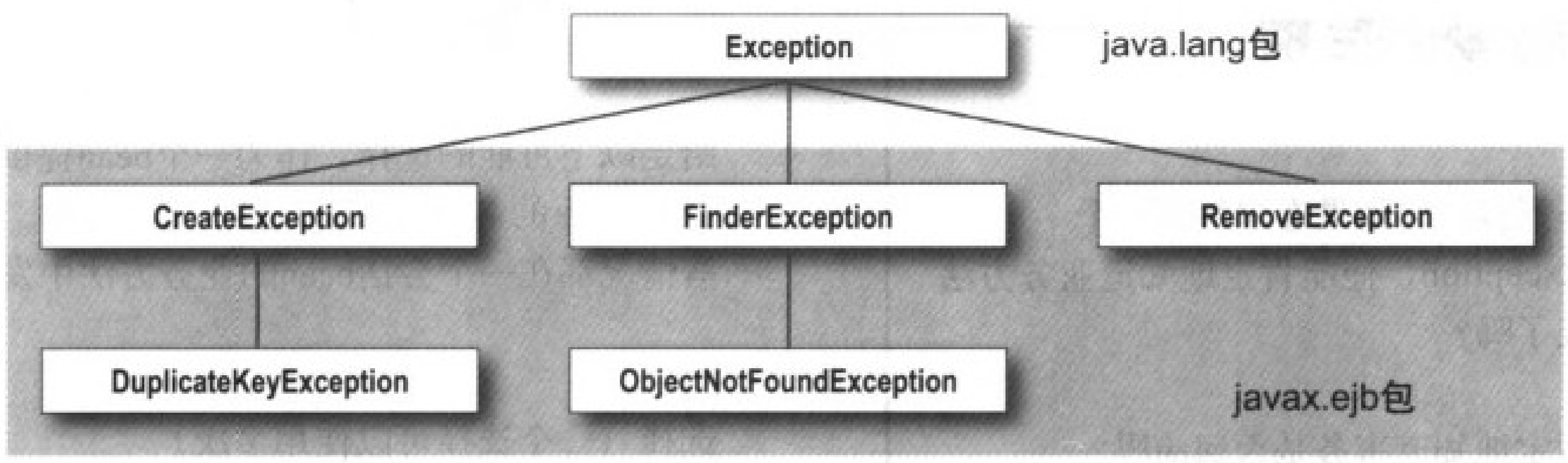
业务逻辑运行时出现一个DivideByZero异常。对此没有一个try/catch。

从ejbCreate()方法抛出一个CreateException，而且你认识到可能无法安全地完成你的事务。

在一个业务方法中捕获到一个受查异常，而且认识到你的bean可能被破坏。

# 5个标准EJB应用异常

javax.ejb包中有EJB容器使用的5个标准应用异常，不过你也可以使用这些异常，可以直接使用，也可以作为定制异常的超类。




**① CreateException**

可以猜到，如果在创建过程中出了问题，容器会从一个创建方法抛出这个异常。这包括运行一个创建方法的时候bean代码自行抛出 CreateException的情况（不过，如果bean要抛出这个异常的话，bean就要在bean类中声明它，而不只是在home接口中声明）。

**② DuplicateKeyException**

如果与创建有关的业务逻辑要求容器使用一个主键来插入一个新实体，但是这个主键已经分配给了另一个实体，此时就会从实体bean创建方法向客户抛出这个异常（可以看出来它是CreateException的一个子类吗?)。

**③ FinderException**

容器从一个实体bean的查找方法向客户抛出这个异常，告诉客户在查找期间出现了问题。bean可以抛出这个异常（只有BMP bean才可以），但是对于CMP bean，只有容器才能抛出这个异常。

**④ ObjectNotFoundException**

如果数据库中没有与查找方法主键参数匹配的实体，容器只会在单实体查找方法中向客户抛出这个异常。

**⑤ RemoveException**

如果删除方法中出现问题，容器会从一个会话bean或实体bean向客户抛出这个异常。不过……也有其他可能，例如，如果bean提供者不想让客户删除实体bean，也可以抛出这个异常。

# 在客户看来的5个标准应用异常

在这5个标准应用异常中，有两个更为特定：DuplicateKeyException和ObjectNotFoundException。如果客户得到一个ObjectNotFoundException，相对于得到一个更抽象的FinderException来说，客户可以获得更多的信息。如果客户得到一个DuplicateKeyException，与得到一个一般的CreateException相比，他会对哪里出了问题有更多的了解。




**CreateException**

客户不能肯定地知道bean是否确实创建。容器可能在事务提交之后出问题。




**DuplicateKeyException**

如果客户捕获到一个DuplicateKeyException他可以完全相信bean没有创建。




**FinderException**

客户不知道数据库中是否存在一个匹配的实体（对于多实体查找方法，则为是否存在一些匹配的实体）。容器可能在查找数据库之前就出了问题，而导致抛出FinderException。




**ObjectNotFoundException**

如果客户捕获到一个ObjectNotFoundException，他可以完全相信数据库中没有与这个主键匹配的实体。要记住，ObjectNotFoundException只针对单实体查找方法，所以客户不会从一个多实体查找方法得到这个异常。




**RemoveException**

客户不能肯定bean是否被删除。bean提供者只是拒绝了客户的请求，如“你可以整天地请求，但是我实在没有办法从数据库删除那个实体”，或者实体可能已经从数据库删除，但是随后出了问题而导致了RemoveException。

**无状态会话bean客户不会得到RemoveException!**

如果看到这样一种情况，客户得到了一个RemoveException，那你就完全能排除无状态会话bean的可能性。想想看……无状态会话bean的删除与客户没有关系，所以没有人能给出这个异常。可以从ejbRemove()方法抛出一个RemoveException，但是如果bean是一个无状态会话bean，客户永远也不会看到这个异常。

**客户可能得不到最特定的异常。**

不要指望客户总能得到最特定的异常。根据规范，即使问题是由于重复主键造成的，容器也可能不向客户发出一个DuplicateKeyException。

这对客户来说意味着什么？如果客户的代码捕获了CreateException和DuplicateKeyException，倘若捕获到一个CreateException，这个问题也有可能是由于重复主键造成的；也就是说，尽管捕获到的是CreateException而非DuplicateKeyException，但并不能说这个问题肯定不是重复主键导致的。

例如，如果客户代码只是一味地尝试create()，但却只提供一个主键，心里想“我得到的又不是DuplicateKeyException，肯定不是重复主键的问题……”，这就错了。如果客户得到一个CreateException，而不是DuplicateKeyException，你并不能知道键是否合适。到底容器会不会给你一个更特定的异常，这要由容器决定。

## there are no Dumb Questions

**问：** **既然RemoveException对无状态会话bean没有意义，为什么还允许抛出这个异常呢（假设在bean的remove()方法中声明了这个异常）？**

**答：** 记住，bean类中没有什么能指出这个bean到底是不是无状态的。当然，如果一个bean是有状态bean，这是可以分辨出来的，因为只要有重载的创建方法（或者创建方法有参数），这样的bean肯定就是有状态bean。但是，即使一个会话bean只有一个无参数的创建方法，也不能确定这个bean在部署时会标记为无状态还是有状态。

如果单独为StatelessSessionBean和StatefulSessionBean有不同的接口，情况就不同了，这样有状态和无状态会话bean要分别实现不同的接口（而不是同一个SessionBean），但是现在还没有这样区分开。

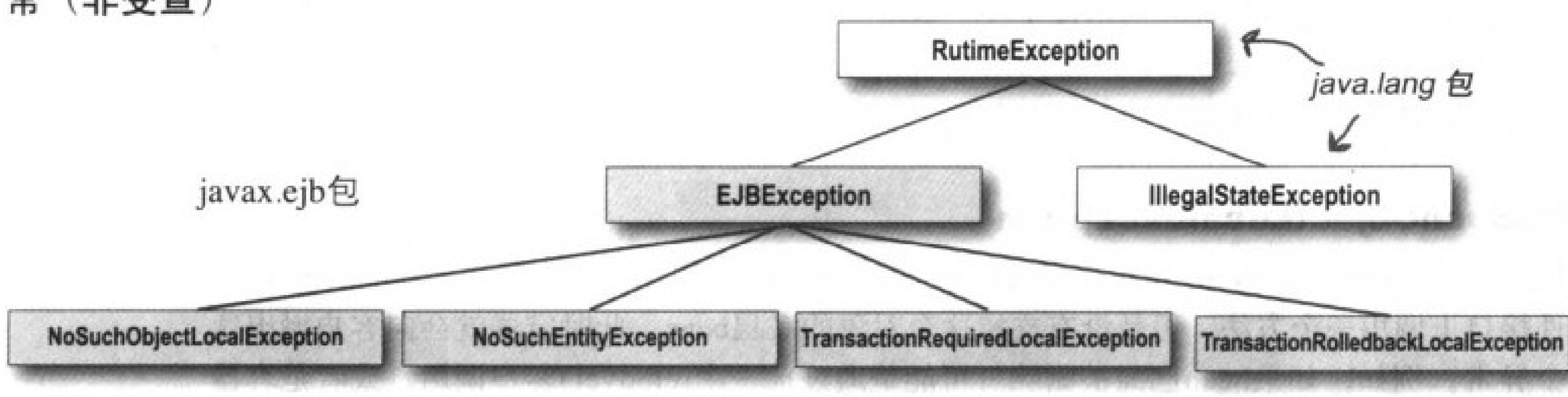
**问：** **如果bean提供者不想让实体bean客户删除一个bean，为什么还有一个删除方法呢？**

**答：** 你是不是忘了这里有一个很明显的东西？remove()方法放在哪里？它们是怎么展现给客户的？想起来了吧。对，就是放在客户接口里。要记住，对于一个远程实体bean客户，有3个可用的删除方法（无参数的删除方法在EJBObject接口中，取一个主键的删除方法和取一个句柄的删除方法都在EJBHome接口中）。本地实体bean客户有两个删除方法，一个在EJBObject接口，另一个取主键作为参数，放在home接口中。所以要想对客户隐藏remove()方法是做不到的，不过，如果你的bean不支持删除，可以从bean的ejbRemove()方法抛出一个RemoveException，这样容器不会完成删除。这就是在威胁容器，“别这么做！！不管客户怎么说，看你敢做这个删除！”

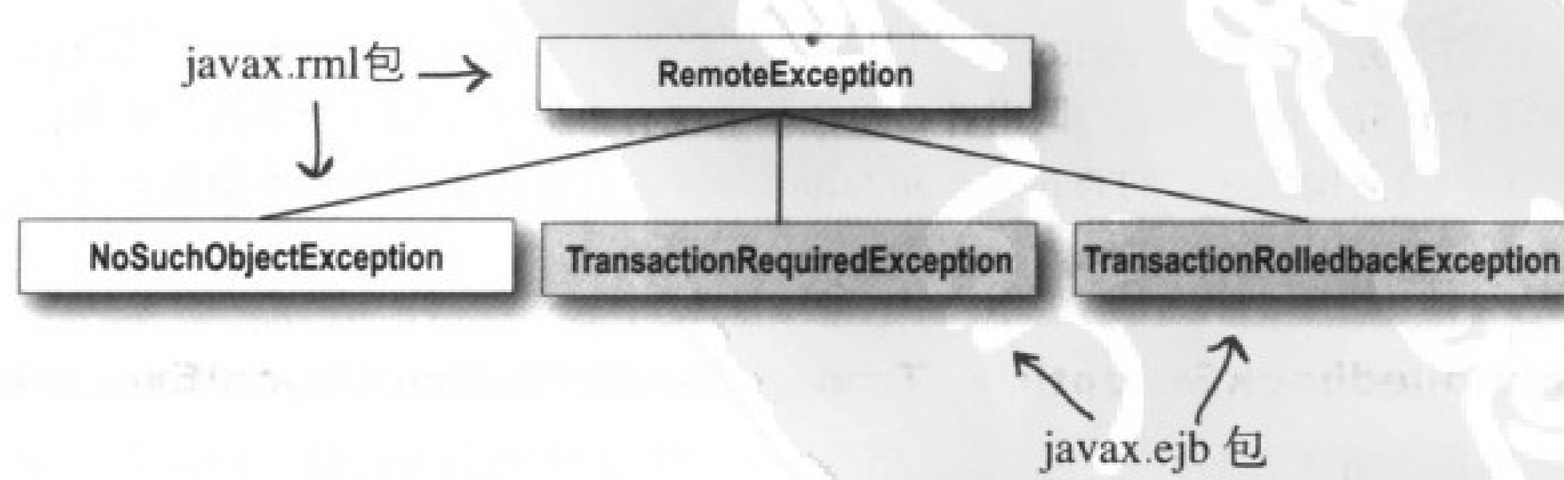
# 常用的系统异常

除了标准的EJB应用异常，还有一些很重要的系统异常。其中一些（包括EJBException）是J2EE API的一部分，这里最常用的EJB系统异常要算java.rmi.NoSuchObjectException和java.lang.IllegalStateException。

### 本地客户和bean会得到这些异常（非受查）




### 只有远程客户会得到这些异常（受查）



# 常用的系统异常

### ① IllegalStateException

如果bean在其上下文上调用了一个方法，但此时还不允许这样做，容器就会向这个bean抛出这个异常。例如，会话bean在setSessionContext()方法中不能向它的上下文要其EJB对象的一个引用。这个时候还太早。如果bean在没有事务的时候调用一个事务方法，如setRollbackOnly()或getRollbackOnly()，也会得到一个IllegalStateException！

### ② EJBException

bean抛出这个异常，告诉容器出现了一个系统异常（这要求容器回滚事务，将异常记入日志，杀死bean，并给本地客户一个EJBException，而给远程客户一个RemoteException）。不过，容器还可能因为其他许多原因抛出这个异常，这些原因稍后再做说明。

### ③ NoSuchObjectLocalException

这个异常是java.rmi.NoSuchObjectException的“本地版本”。当客户在一个本地home或组件接口上调用一个方法，但是没有支持这个对象的底层bean，此时容器就会向客户抛出这个异常。例如，如果bean已经被删除（可能是通过前一个remove()客户调用，或者是容器由于一个异常、有状态bean超时，或者要缩减bean池的大小等等原因，而杀死了bean），就可能发生这种情况。

### ④ NoSuchEntityException

这个异常可能见得不多，你可能也不会用得太多，特别是CMP，但是如果你想告诉容器你要访问的实体不再位于数据库中（可能是因为被一个管理应用删除了），就需要从bean代码抛出这个异常。

### ⑤ TransactionRequiredLocalException / TransactionRequiredException

如果被调用方法的事务属性为Mandatory，但是这个调用没有事务上下文，此时容器就会向调用者抛出这个异常。换句话说，调用者必须在一个现有事务中调用这个方法，如果没有这样一个事务，就会抛出适当的TransactionRequired异常（取决于客户是本地客户还是远程客户）。

### ⑥ TransactionRolledbackException / TransactionRolledbackLocalException

容器无法提交事务，而且调用者是在一个现有的事务上下文中调用方法时，容器会向调用者抛出这个异常。不过，如果无法提交的原因是bean明确地调用了setRollbackOnly()方法，那么容器不会抛出这个异常。在这种情况下，容器会回滚事务，把业务方法结果传回客户（除非bean也抛出一个应用异常）。




受查
java.rmi. NoSuchObjectException

只有这个异常不在javax.ejb而在java.rmi包中。这说明客户有一个桩，但是远程对象没有了。

非受查
javax.ejb.NoSuchObjectLocalException

这是EJB 2.0增加本地接口时增加的。这表示客户有一个无效的EJB对象引用。

受查
javax.ejb.ObjectNotFoundException

这是惟一的用于查找方法的异常！（单实体查找方法）

非受查
javax.ejb.NoSuchEntityException

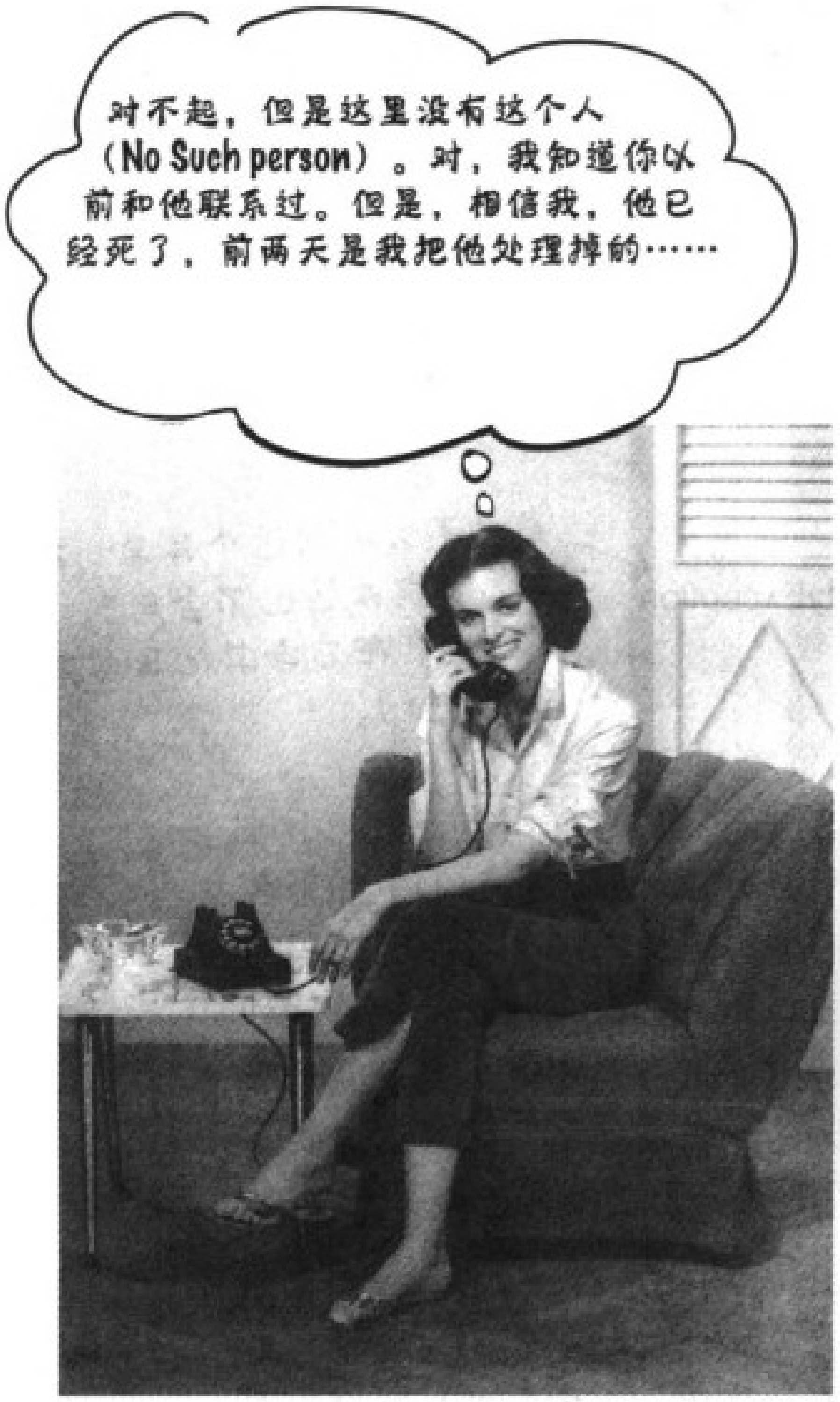
客户不会看到这个异常，而且你可能永远也不会用到它……别把它与其他异常混淆就行了！

对，我们也承认，这些异常名是有一些随意。不过要记住，是先有NoSuchObjectException的。如果客户有一个远程对象的桩，但是出于某种原因远程对象不再可用，NoSuchObjectException就是针对这种情况设计的。可能是由于服务器瘫痪，也可能是服务本身没有解除部署，到底是什么原因并不重要。对客户而言，总之是与服务器端断开连接了。

NoSuchObjectLocalException是EJB 2.0中增加本地客户视图时增加的。这个概念稍有不同，因为本地客户不使用桩，而是使用一个实际的Java对象引用。但是从客户的角度看，到底是本地还是远程没有什么关系，NoSuchObject<whatever>异常意味着你不能再使用EJB引用来得到bean了！必须回到home重新再查一遍。




Object**NotFound**Exception 是一个**Finder**Exception




**NoSuch**ObjectException是指，客户还有一个远程对象的桩，但是这个对象已经没有了（或者出于某种原因不再有效）。

**NoSuch**ObjectLocalException是指，一个本地客户有EJB对象的一个引用，但是这个EJB对象已经被删除（或者出于某种原因不再有效）。

## 要点

- Java中的异常可以是受查异常也可以是非受查异常。受查异常要扩展java.lang.Exception，但不能扩展java.lang.RuntimeException。
- EJB中的异常可以从bean抛给容器，也可以从容器抛给bean。
- EJB异常有两种类型：应用异常和系统异常。
- 应用异常是客户预期的受查异常，而且可以从中恢复。这包括所有受查异常，但java.rmi.RemoteException和它的子类除外。
- 系统异常就是所有其他的异常，包括所有运行时异常，以及RemoteException。
- 对于客户来说，可以把RemoteException看作是一个发生在服务器上的运行时异常（也就是说，某种未预期的异常）。
- 容器得到一个系统异常时，它会回滚事务，把异常记入日志，并丢掉（杀死）bean。
- 容器得到一个应用异常时，它会照接收时的原样把异常发送给客户。事务不会自动回滚，bean的生命还维持着。
- bean提供者应该原样把应用异常交给容器，这样容器可以把这些异常传给客户。如果bean的事务可能被破坏，bean提供者应当在抛出应用异常之前调用setRollbackOnly()，这样容器就不会提交事务了。
- 如果bean提供者捕获到一个异常，而且无法从中恢复，他要把这个异常包装为一个EJBException并重新抛出（这是一个运行时异常，所以无需声明）。
- 有5个标准EJB应用异常：CreateException、DuplicateKeyException（扩展了CreateException）、FinderException、ObjectNotFoundException（扩展了FinderException）和 RemoveException。
- 常用的系统异常包括EJBException、IllegalStateException、TransactionRequiredLocalException和NoSuchObjectLocalException。
- 当单实体查找方法无法在持久存储库中找到一个匹配实体，即与查找方法的主键参数匹配的实体，此时就要使用ObjectNotFoundException。
- NoSuchObjectException是一个RMI异常，客户有一个远程对象的桩，但是这个远程对象已经被删除，或者出于某种原因不再有效，此时就要使用NoSuchObjectException。
- NoSuchObjectLocalException是一个运行时异常，一个本地客户有EJB对象的一个引用，但是这个EJB对象不再有效（很有可能是因为bean已经被删除），此时就要使用NoSuchObjectLocalException。

# 场景：你认为会发生什么？

从理论上讲，这一部分就像是一个隐含的练习，而且这个题目比较大。不过我们已经告诉你这是一个练习，所以不算是秘密了。当然，我们也可以逐项地把每一个问题的答案都告诉你……但是如果你能自己做出来，肯定能记得更牢。在这一章的最后部分我们会全面总结，但是现在先不要直接去看！就算你从来不做练习，也要把这些场景走一遍。等你通过了认证考试，拿到那个可爱的小胸针的时候，你会为此对我们感激不尽的。

1. 消息驱动bean的onMessage()方法捕获到一个应用异常。它能把这个应用异常重新抛给容器吗？

   （提示：容器从bean得到一个应用异常时通常是怎么做的？在这种情况下容器能那么做吗？）

2. 一个使用CMT的会话bean有一个事务属性标记为NotSupported的方法。在这个方法运行时，bean在它的上下文上调用了setRollbackOnly()。这会导致一个异常吗？是个什么样的异常？

   （提示：考虑一下setRollbackOnly()会做什么，还有bean必须在什么状态。）

3. 一个消息驱动bean在onMessage()方法中调用了getCallerPrincipal()，会发生什么？

   （提示：getCallerPrincipal()是用来干什么的？在这里它有意义吗？）

4. 一个使用CMT的会话bean有一个事务属性标记为Mandatory的方法。调用这个方法的客户不在一个事务中，会发生什么呢？

   （提示，想想常见系统异常的名字。有没有哪个异常对应这里的情况？）

5. bean发现它不能提交一个事务，但是又不希望客户得到一个异常，这个bean该怎么做呢？

6. 一个bean希望客户得到一个应用异常，但是仍希望事务能提交，它该怎么做呢？

**Sharpen your pencil**

把场景与可能出现在这种情况下的异常匹配起来。先不要看下一页！！下一页就是答案。

| 场景 | 异常 |
| --- | --- |
| 一个CMT bean调用context.getUserTransaction()。 | IllegalStateException |
| 客户在已经删除的bean上调用remove()。 | EJBException |
| 客户在一个有状态bean上调用remove()，这个bean还在一个打开的事务中。 | CreateException |
| 会话bean在它的上下文上调用getPrimaryKey()。 | RemoveException |
| 一个CMT bean从一个标记为NotSupported的方法调用getRollbackOnly()。 | ObjectNotFoundException |
| 客户在一个远程CMT bean上调用一个方法，而且这个方法标记为Mandatory。调用者在调用方法时没有一个事务上下文。 | NoSuchObjectException |
| 一个消息驱动bean从onMessage()方法在bean上下文上调用isCallerInRole()。 | RemoteException |
| 客户在一个会话bean的本地home接口上调用home删除方法。 | TransactionRequiredException |
| 无状态会话bean从setSessionContext方法在bean上下文上调用getCallerPrincipal()。 | NotSupportedException |
| 客户Foo调用一个远程有状态bean上的一个方法，而这个bean已经在执行客户Bar的一个方法。 | RuntimeException |
| 客户调用了一个会话bean本地组件接口上的getPrimaryKey()。 | NoSuchObjectLocalException |
| 消息驱动bean捕获到一个NamingException，而且无法从中恢复。bean可以抛出什么异常（如果要抛出）来告诉容器？ | DuplicateKeyException |
| 会话bean希望容器知道事务要回滚，而且bean应当被杀死。 | SystemException |
| 客户调用findByPrimaryKey（“23”），而此时没有一个主键为“23”的实体。 | NoSuchEntityException |
| 尽管bean一切正常，但容器有一个系统异常，想把它抛给一个本地客户。 | |

# 场景小结

## 事物场景

**一个CMT bean调用context.getUserTransaction()。**

这个bean会得到一个IllegalStateException（只有BMT bean能得到一个UserTransaction）。

**客户在一个有状态bean上调用remove()，这个bean还在一个打开的事务中。**

客户得到一个RemoveException。如果一个有状态bean还在事务当中，就不能将其删除。（这很好地说明了为什么让一个事务在多个客户调用期间一直打开是种不好的做法。换句话说，如果你在一个方法中开始了一个BMT事务，就应该在该方法结束前结束这个事务！）

**一个CMT bean从一个标记为NotSupported的方法调用getRollbackOnly()。**

bean会得到一个IllegalStateException。调用getRollbackOnly()或setRollbackOnly()时必须在一个事务中。这说明，如果一个方法标记为NotSupported、Never或Supports（会话bean和实体bean）或NotSupported（消息驱动bean，记住，消息驱动bean不能使用Never或Supports，因为事务永远也不会传播到一个消息驱动bean中去，所以这两个属性是没有意义的），那么在这个方法中就不能调用getRollbackOnly()或setRollbackOnly()。

**客户在一个远程CMT bean上调用一个方法，而且这个方法标记为Mandatory。调用者在调用方法时没有一个事务上下文。**

客户得到一个TransactionRequiredException（本地客户会得到TransactionRequiredLocalException）。

**会话bean希望容器知道事务要回滚，而且bean应当被杀死。**

bean应当抛出一个EJBException。容器接管过去，一如平常地对付这个系统异常，回滚事务，把异常记入日志，杀死bean，向远程客户抛出一个RemoteException，或者向本地客户抛出一个EJBException。

# 场景小结

## 客户场景

**客户在已经删除的bean上调用remove()。**

你可能想客户会得到RemoveException，但并不是这样！要记住，remove()也只是bean接口中的一个方法，如果在一个已经删除的bean上调用这个方法，和在已删除的bean上调用任何其他业务方法一样，你会得到同样的异常，如果是远程客户，会得RemoteException，如果是本地客户，则会得到EJBException。

**客户在一个会话bean的本地home接口上调用home删除方法。**

客户得到一个RemoveException。为什么呢？因为会话bean本地home接口上的惟一的一个remove()方法是取一个主键作为参数的删除方法，而对于会话bean这是不能用的。远程客户也会得到一个 RemoveException。

**客户Foo调用一个远程有状态bean上的一个方法，而这个bean已经在执行客户Bar的一个方法。**

客户Foo得到一个RemoteException（如果客户Foo是本地客户，他会得到一个EJBException）。会话bean一次只能处理一个客户！

**客户调用了一个会话bean本地组件接口上的getPrimaryKey()。**

这与客户调用一个本地bean home上的remove()很类似。客户会得到一个EJBException（如果是远程客户，他会得到一个RemoteException）。 为什么呢？会话bean没有主键！如果bean是一个会话bean，只要是与这个bean主键有关的方法调用都将失败。

**客户调用findByPrimaryKey（“23”），而此时没有一个主键为“23”的实体。**

客户得到一个ObjectNotFoundException，这是FinderException的一个子类。

**尽管bean一切正常，但容器有一个系统异常，想把它抛给一个本地客户。**

客户得到一个EJBException。

# 场景小结

## bean场景

**会话bean在它的上下文上调用getPrimaryKey()。**

bean得到一个IllegalStateException。原因你应该知道，会话bean是没有主键的……

**一个消息驱动bean从onMessage()方法在bean上下文上调用isCallerInRole()。**

想想看。消息驱动bean有调用客户吗（我们不认为容器是一个"客户"，尽管容器会调用消息驱动bean的方法）。如果没有客户，那么bean能得到谁的安全信息呢？bean为此会得到一个IllegalStateException。

**无状态会话bean从setSessionContext方法在bean上下文上调用getCallerPrincipal()。**

bean会得到一个IllegalStateException，因为setSessionContext()在bean的一生中为时过早，还不能得到客户的信息。实际上，无状态bean只能在运行组件接口的一个业务方法时才能得到客户的安全信息。而且，即使bean是有状态的，要在setSessionContext中得到客户信息也太早了，尽管有状态bean可以在ejbCreate()中调用getCallerPrincipal()和isCallerInRole()（但无状态bean不行）。

**消息驱动bean捕获到一个NamingException，而且无法从中恢复。bean可以抛出什么异常（如果要抛出）来告诉容器？**

bean应当抛出一个EJBException。要记住，消息驱动bean没有客户。它们没有客户接口，所以没有地方声明应用异常。这意味着消息驱动bean只能抛出系统异常。消息驱动bean能抛出的惟一的异常就是EJBException。它是不能抛出应用异常的，而且应该把其他系统异常传播给容器。




填表，描述远程客户和本地客户的差别。 这个表可以拿来备考，所以一定要重视。

| 远程客户和本地客户有些方面是一样的，不过在这里我们不做说明。 | 远程客户 | 本地客户 |
| --- | --- | --- |
| bean中的系统异常如何传递到客户。 | | |
| 客户调用一个已删除的bean上的方法时会得到什么异常。 | | |
| 客户在没有事务上下文的情况下调用一个标记为Mandatory的方法时会得到什么异常。 | | |
| 客户开始一个事务，而容器必须回滚事务的时候，会有什么异常。 | | |
| 客户在会话bean组件接口上调用getPrimaryKey()时会得到什么异常。 | | |
| 客户调用一个会话bean上的方法，而这个bean已经在为另一个客户执行一个方法，此时会得到什么异常。 | | |
| 客户调用一个有状态会话bean的remove()方法，而这个bean仍在一个事务中，此时会得到什么异常。 | | |




好像见过？不错，在这一章刚开始的时候我们见过这个表。只不过那时表已经填好。现在该轮到你来填了。如果除了文字说明还有图表描述，可以加分。

| | 应用异常 | 系统异常 |
|---|---|---|
| 客户恢复 | | |
| 事务状态 | | |
| bean实例 | | |
| 日志记录 | | |
| 示例 | | |
| 编译器检查和其他规则 | | |

# COFFEE CRAM
# 模拟测验

**1** 关于实体bean客户什么时候收到一个javax.ejb.EJBException，下面哪种说法是正确的？（选出所有正确的答案。）

- ❑ A. 客户必须是远程客户
- ❑ B. 客户必须是本地客户
- ❑ C. 客户永远不会收到这个异常
- ❑ D. 客户必须处理或声明这个异常
- ❑ E. 只有当客户在一个事务中时才会出现这个异常

**2** 哪种情况会导致抛出一个java.rmi.NoSuchObjectException？（选出所有正确的答案。）

- ❑ A. 远程客户调用一个已经删除的有状态会话bean上的方法
- ❑ B. 远程客户调用一个已经删除的实体bean上的方法
- ❑ C. 远程客户调用一个查找方法，但参数不合法
- ❑ D. 容器在一个没有准备好串行化的bean上调用ejbPassivate()方法

**3** 以下哪一个是java.lang.RuntimeException的子类？（选出所有正确的答案。）

- ❑ A. **`javax.ejb.EJBException`**
- ❑ B. **`javax.ejb.RemoveException`**
- ❑ C. **`javax.ejb.CreateException`**
- ❑ D. **`javax.ejb.NoSuchEntityException`**
- ❑ E. **`java.rmi.RemoteException`**
- ❑ F. **`javax.ejb.ObjectNotFoundException`**
- ❑ G. **`java.rmi.NoSuchObjectException`**

**4** 以下哪一个是EJB 2.0规范中有关系统异常的原则？（选出所有正确的答案。）

- ❏ A. bean方法应当捕获RuntimeException异常
- ❏ B. bean方法应当把不可恢复的受查异常包装在一个**`javax.ejb.EJBException`**异常中
- ❏ C. 对于远程客户，bean方法应当把不可恢复的受查异常包装在一个**`java.rmi.RemoteException`**中
- ❏ D. 如果一个CMT bean抛出一个系统异常，事务仍会提交，除非bean调用了**`setRollbackOnly`**

**5** 如果一个CMT划分bean的业务方法抛出一个系统异常，在哪种情况下容器总会抛出一个 **`javax.transaction.TransactionRolledbackException`**?（选出所有正确的答案。）

- ❏ A. 如果这个方法的事务属性标记为“RequiresNew”
- ❏ B. 如果这个方法的事务属性标记为 “Mandatory”
- ❏ C. 如果这个方法的事务属性标记为 “Never”
- ❏ D. 如果这个方法的事务属性标记为 “NotSupported”

**6** 以下哪一个是javax.ejb.FinderException的子类？（选出所有正确的答案。）

- ❏ A. CreateException
- ❏ B. NoSuchEntityException
- ❏ C. RemoveException
- ❏ D. DuplicateKeyException
- ❏ E. ObjectNotFoundException

**7** 有CMT划分的MDB可以从哪个方法抛出应用异常？（选出所有正确的答案。）

- ❏ A. **`onMessage()`**
- ❏ B. **`ejbCreate()`**
- ❏ C. **`ejbRemove()`**
- ❏ D. **`getUserTransaction()`**
- ❏ E. **`setMessageDrivenContext()`**
- ❏ F. 以上都不行

**8** 如果一个本地客户收到来自EJB调用的异常，以下哪一种说法是正确的？（选出所有正确的答案。）

- ❑ A. 这个异常必须是java.rmi包中的异常
- ❑ B. 这个异常可能是一个应用异常
- ❑ C. 这个异常可能是一个系统异常
- ❑ D. 以上都不对

**9** 如果有BMT划分的一个消息驱动bean抛出一个系统异常，容器会采取什么动作？(选出所有正确的答案。)

- ❑ A. 记录异常日志
- ❑ B. 删除实例
- ❑ C. 标记事务回滚
- ❑ D. 提交事务，除非bean调用了`setRollbackOnly()`
- ❑ E. 以上都不对

**10** 客户可以从哪种类型的bean收到系统异常？（选出所有正确的答案。）

- ❑ A. 有CMT划分的会话bean
- ❑ B. 有BMT划分的会话bean
- ❑ C. 有CMT划分的消息驱动bean
- ❑ D. 有BMT划分的消息驱动bean
- ❑ E. 有CMT划分的实体bean

# COFFEE CRAM 模拟测验答案

**1** (规范：374)

关于实体bean客户什么时候收到一个javax.ejb.EJBException，下面哪种说法是正确的？（选出所有正确的答案。）

- ❑ A. 客户必须是远程客户 — 远程客户得到的是RemoteException
- ☑ B. 客户必须是本地客户
- ❑ C. 客户永远不会收到这个异常
- ❑ D. 客户必须处理或声明这个异常
- ❑ E. 只有当客户在一个事务中时才会出现这个异常

**2** (规范：374)

哪种情况会导致抛出一个java.rmi.NoSuchObjectException？（选出所有正确的答案。）

- ☑ A. 远程客户调用一个已经删除的有状态会话bean上的方法
- ☑ B. 远程客户调用一个已经删除的实体bean上的方法
- ❑ C. 远程客户调用一个查找方法，但参数不合法 — 客户得到FinderException或ObjectNotFoundException（FinderException的子类）
- ❑ D. 容器在一个没有准备好串行化的bean上调用ejbPassivate()方法

**3** (API文档)

以下哪一个是java.lang.RuntimeException的子类？（选出所有正确的答案。）

- ☑ A. `javax.ejb.EJBException`
- ❑ B. `javax.ejb.RemoveException`
- ❑ C. `javax.ejb.CreateException`
- ☑ D. `javax.ejb.NoSuchEntityException`
- ❑ E. `java.rmi.RemoteException`
- ❑ F. `javax.ejb.ObjectNotFoundException`
- ❑ G. `java.rmi.NoSuchObjectException`

**4** 以下哪一个是EJB 2.0规范中有关系统异常的原则？（选出所有正确的答案。）（规范：373～374）

- ❑ A. bean方法应当捕获RuntimeException异常
- ☑ B. bean方法应当把不可恢复的受查异常包装在一个**`javax.ejb.EJBException`**异常中
- ❑ C. 对于远程客户，bean方法应当把不可恢复的受查异常包装在一个**`java.rmi.RemoteException`**中 —— 不，容器会做这个工作
- ❑ D. 如果一个CMT bean抛出一个系统异常，事务仍会提交，除非bean调用了**`setRollbackOnly`** —— 系统异常总是会导致事务回滚

**5** 如果一个CMT划分bean的业务方法抛出一个系统异常，在哪种情况下容器总会抛出一个 **`javax.transaction.TransactionRolledbackException`**？（选出所有正确的答案。）（规范：375～376）

- ❑ A. 如果这个方法的事务属性标记为“RequiresNew”
- ☑ B. 如果这个方法的事务属性标记为 “Mandatory”
- ❑ C. 如果这个方法的事务属性标记为 “Never”
- ❑ D. 如果这个方法的事务属性标记为 “NotSupported”

其他的选择都是错的，因为它们指示了出现异常时调用者的事务不是当前活动事务。除非回滚的是调用者的事务，否则没有必要（通过异常）告诉容器！

**6** 以下哪一个是javax.ejb.FinderException的子类？（选出所有正确的答案。）（规范：263～264）

- ❑ A. CreateException
- ❑ B. NoSuchEntityException
- ❑ C. RemoveException
- ❑ D. DuplicateKeyException — CreateException的子类
- ☑ E. ObjectNotFoundException

**7** 有CMT划分的MDB可以从哪个方法抛出应用异常？（选出所有正确的答案。）（规范：377）

- ❑ A. **`onMessage()`**
- ❑ B. **`ejbCreate()`**
- ❑ C. **`ejbRemove()`**
- ❑ D. **`getUserTransaction()`**
- ❑ E. **`setMessageDrivenContext()`**
- ☑ F. 以上都不行

— MDB不能声明/抛出任何应用异常……否则，谁来捕获呢？MDB没有客户。

**8** 如果一个本地客户收到来自EJB调用的异常，以下哪一种说法是正确的？（选出所有正确的答案。） （规范：381～382）

- ❑ A. 这个异常必须是java.rmi包中的异常
- ✔ B. 这个异常可能是一个应用异常
- ✔ C. 这个异常可能是一个系统异常
- ❑ D. 以上都不对

**9** 如果有BMT划分的一个消息驱动bean抛出一个系统异常，容器会采取什么动作？（选出所有正确的答案。） （规范：378）

- ✔ A. 记录异常日志
- ✔ B. 删除实例
- ✔ C. 标记事务回滚
- ❑ D. 提交事务，除非bean调用了**`setRollbackOnly()`**
- ❑ E. 以上都不对

**10** 客户可以从哪种类型的bean收到系统异常？（选出所有正确的答案。） （规范：377）

- ✔ A. 有CMT划分的会话bean
- ✔ B. 有BMT划分的会话bean
- ❑ C. 有CMT划分的消息驱动bean
- ❑ D. 有BMT划分的消息驱动bean — MDB没有客户！
- ✔ E. 有CMT划分的实体bean

# 11 EJB中的安全

# 保护你的秘密



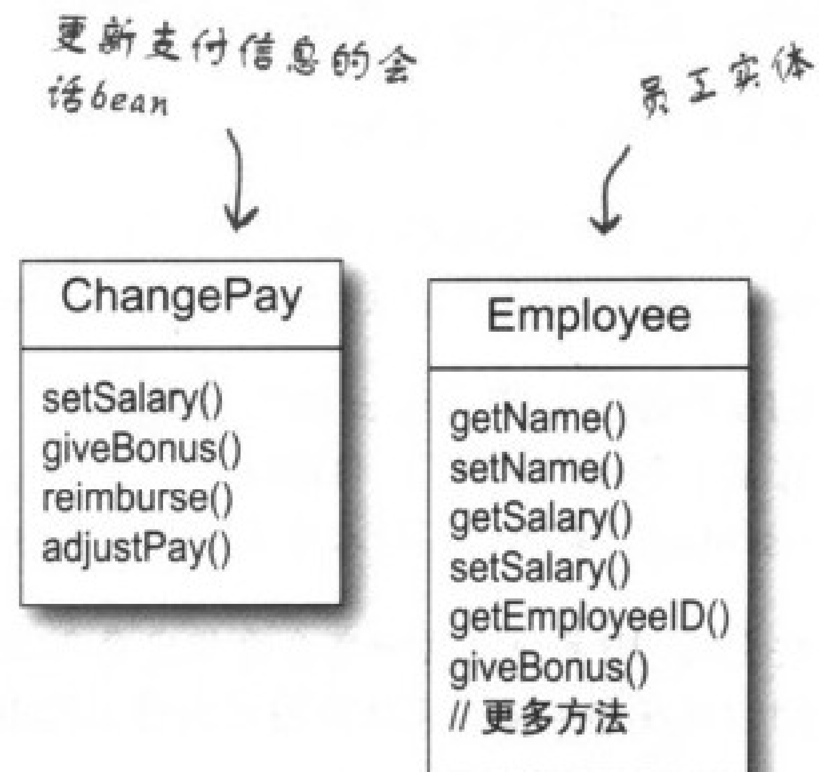

**守住你的秘密。** 安全包括认证和授权。首先，你必须证明你的身分，然后我们会告诉你可以做哪些事情。在EJB中，安全很简单，因为你只需要处理授权，确定谁能调用bean的哪些方法。不过有一个问题要注意……如果你是一个bean提供者或者应用组装人员，你可能不知道最终的用户到底是什么人！所以要准备好，需要建立角色，这就像是工作岗位一样，包括Manager（经理）、Supervisor（主管）、Admin（管理员）等等，当一个实际公司中的某个人部署你的应用时，部署人员会在你“自创”的这些角色名（如Manager）和使用应用的真正的人之间建立映射。

# 安全

| 标准大纲： | | 具体含义： |
| --- | --- | --- |
| 14.1 | 关于EJB对安全管理的支持，包括安全角色、安全角色引用和方法许可，明确哪些说法是正确的，哪些不正确。 | 必须知道EJB中可以实现两种安全：通过程序方式实现，以及通过声明方式实现。声明方式实现是指在部署描述文件中定义安全。需要知道，EJB中的安全主要是关于声明一组抽象安全角色，然后声明这些角色分别可以调用哪些方法。 |
| 14.2 | 给定一组职责，明确其中应用组装人员、bean提供者、部署人员、容器提供者或系统管理员的职责分别是哪些。 | 这一章中要特别注意这一点。要以一种基于组件的开发方法来考虑整个过程，需要了解谁负责什么。你要知道，bean提供者通常没有什么安全职责，但是如果他使用了isCallerInRole()方法，就必须在部署描述文件中放一个安全角色引用，告诉应用组装人员他硬编码写到程序里的角色名（作为isCallerInRole()方法的参数）。你还要知道，应用组装人员的任务是定义安全角色和方法许可，以及定义角色链接（从程序员自造的角色引用映射到应用组装人员建立的实际安全角色）。最后一点，你必须知道，部署人员要负责建立实际用户及实际用户组与应用组装人员的抽象安全角色之间的映射（以一种特定于开发商的方式），这是利用容器工具或性质文件完成的。 |
| 14.3 & 14.4 | 给定一个代码清单，确定它是否可以合法而且/或者适当地采用程序方式访问一个调用者的安全上下文。<br><br>给定一个与安全有关的部署描述文件标记，明确有关该标记的说法和/或代码哪些是正确的，哪些不正确。 | 你必须很清楚，有关安全的部署描述文件标记（包括安全角色引用和角色链接）都在部署描述文件的<enterprise-bean>部分中，而安全角色和方法许可都在<application-assembly>部分中。你必须知道如何使用<security-identity>标记，而且要知道消息驱动bean不能使用<use-caller-identity>!应该知道，只能在一个会话bean或实体bean的上下文上调用两个有关安全的方法（getCallerPrincipal()和isCallerInRole()）。 |

# 假设你在编写一个工资应用……






# 你能做什么？

到底是谁负责保证程序安全？如何做到？关于EJB安全有好消息，也有坏消息。

## ① EJB中的安全很容易。仅与授权相关。

- 大多数情况下，不用在bean类中放任何有关安全的代码。
- 实际上，bean提供者甚至不用考虑安全，除了在非常特殊的情况下才有必要考虑安全，我们将在本章最后一部分讨论这些情况。
- EJB中的安全是要指出谁能调用什么。可以限制对应用中各个方法的访问，有某个特权“角色”［Director（主任）、Payroll Admin（管理员）、Payroll Assistant（助手）］的人才能访问某些方法。
- 安全的EJB部分（这一部分要指定角色以及这些角色可以访问的方法）是采用声明方式在部署描述文件中使用简单的XML标记完成的。
- 在EJB层次上，可以简单地记为“setSalary()方法只能由主任和工资管理员角色的人访问”。

## ② EJB中的安全是抽象的。它不涉及认证。

- 根据EJB规范，定义角色很容易，为方法指定角色从而控制对这些方法的访问也很简单。但是，系统怎么知道哪个实际的人属于哪个角色呢？对此规范没有做任何定义。要在EJB部署描述文件之外（当然这也在规范之外）指出Jack O Bryan的角色是Director（还可能是其他一些角色）。或者说，XYZ公司的所有Payroll Manager（经理）都有应用中所标的Payroll Admin（管理员）角色。
- 规范中也没有定义如何向系统完成一个实际的人的认证。换句话说，你怎么知道一个人的实际身分，比如说，以Jack O Bryan身分登录进来的人真是Jack O Bryan吗？对此EJB中没有任何说明。完成认证可以有很多种方式，不过其中最典型的是通过支持J2EE的Web服务器，在Web层（客户使用浏览器登录）完成客户认证。
- 实际操作环境（运行应用的公司）的安全必须有一个安全架构，而且要有合适的格式，以便特定的EJB服务器能够与之结合。或者，至少要有一个适当的服务器，允许你配置一个特定的安全领域。就算只是一组名字、口令以及这些名字所属的“角色”，不管怎样，这样一种EJB之外的基础架构还是很需要的。

# EJB中如何做到安全

我的任务很容易。大多数情况下，在我编写bean的时候甚至不用考虑安全。这样很好，因为我的处世哲学就是“安全很难…… 不要管安全。”



Bill
bean提供者

我的任务更复杂一些，我要确定应用中有哪些角色。对于Bill的工资bean（以及我自己的一些bean，这些bean是为了构成一个完整的应用而增加的），我选择了这样一些角色：Payroll Admin（管理员）、Payroll Assistant（助手）和Payroll Director（主任）。然后我在部署描述文件中建立方法许可，指出哪些角色可以调用哪些方法。



Annie
应用组装人员

Annie可以随心所欲地建立很多角色……但是除非我为这些角色映射实际的人或用户组，否则这些角色是没有意义的，因为没有人能调用这些方法。我必须适当地配置服务器，让它使用我们内部员工系统中的用户和用户组，而且我要建立实际用户组与Annie定义的角色之间的映射。让我看看……我要把我们的Payroll Manager（经理）组映射到Annie所说的“Payroll Admin”（管理员）角色……



Dick
部署人员

# 应用组装人员的任务：访问控制



应用组装人员很了解应用。她知道每个方法是干什么的，还知道这些方法该如何使用。应用组装人员知道对于这个应用有哪些有逻辑意义的抽象角色。例如，她知道工资应用中不需要一个Marketing（市场部门）角色。但是这个应用至少要有3个层次的访问：

(1) 有些人能全权控制，他们不仅能查看一个员工的工资数据，还可以修改。

(2) 有些人可以读所有数据，但只能修改部分数据。

(3) 有些人可以读某些数据，但是什么也不能修改。

所以应用组装人员的任务可以分为两部分：

## ① 定义角色

- 得出应用中有哪些有意义的角色，并提供这些角色的名字。由于应用组装人员可能不在应用实际运行的环境中工作，她只是会自创一些抽象的角色名。换句话说，她给出的名字不一定要与实际的名字对应。她甚至可以把这3个角色命名为Clown、Mime和Juggler，对此我们并不关心。只要能很好地描述角色，让部署人员知道哪些实际的人属于这些角色就可以了，到底是什么名字不重要。

- 在部署描述文件中，为应用中的每个角色定义一个**<security-role>**元素。

- 在部署描述文件中，使用**<role-name>**元素为这个角色定义自造的名字。

## ② 指定方法许可

- 从客户视图得出各个角色应当访问哪些方法。要记住，客户视图不仅包括bean提供者在bean的home接口和组件接口中声明的方法，还包括超接口EJBObject和EJBHome（或EJBLocalObject和EJBLocalHome）的方法。

- 在部署描述文件中，定义一个**<method-permission>**元素，其中列出一个或多个安全角色以及一个或多个方法。

- 如果愿意，应用组装人员可以使用**<unchecked/>**元素来指示某个方法不需要任何授权（换句话说，谁都可以调用这个方法）。

- 如果愿意，应用组装人员可以使用**<exclude-list>**元素来定义谁都不能调用的一组方法。换句话说，任何角色都不能调用这个排除列表中的方法。

## there are no Dumb Questions

**问：你提到了排除列表。如果任何人都不允许调用这个方法，那这个方法还有什么存在必要的呢？**

**答：** 请记住，bean是可重用的组件，它要组装到多个应用当中。在一个应用中有意义的方法到了另一个应用中可能就没有意义了。例如，有一个表示数据库中产品的ProductInventory bean，它可能有一些方法允许目录管理员增加产品或者修改价格等等，但是如果有一个客户目录程序使用了这个bean，这个应用就只需要只读方法（也就是获取方法），其他的方法都不需要。

**问：我觉得编写bean的人肯定知道应该有哪些角色。**

**答：** 记住，bean是可重用组件。不错，这句话在上一个问题中提到过，但它在这里也适用。bean提供者并不知道这些bean会怎样使用。如果在设计代码和编写bean时过于特定，那这个bean可能无法对外使用，而只能在这个特定的工资应用中使用（因为它过分地考虑了这个应用）。

只有应用组装人员才知道应用中的各个部分之间如何关联，以及要有哪些角色。对此有一个小小的例外，我们将在本章后面介绍“程序方式实现安全”时讨论这种情况。

## Sharpen your pencil

请看以下应用，得出对于这些应用，有哪些有意义的安全角色。我们会在后面说明各个角色可以调用哪些方法。

1. 一个在线商店的目录系统，记录已售出和已发货的商品、目录等级和价格。

2. 一个通过旅行社完成度假预订的应用。

3. 一个基于规则的产品推荐系统，向用户询问一系列的问题，然后对用户该买什么产品给出适当的建议。

4. 在线红娘/个人广告服务。



# 定义角色

## <security-role>元素

在部署描述文件的<assembly-descriptor>部分创建安全角色。记住，部署描述文件有两部分：

(1) bean提供者创建的 <enterprise-beans>部分，其中描述了应用中的各个bean。

(2) 应用组装人员创建的<assembly-descriptor>部分，其中描述了应用整体的特征，包括bean如何相互引用、大多数安全信息以及事务属性。

```
<assembly-descriptor>

    <security-role>
        <role-name>Payroll Director</role-name>
    </security-role>

    <security-role>
        <role-name>Payroll Admin</role-name>
    </security-role>

    <security-role>
        <role-name>Payroll Assistant</role-name>
    </security-role>
...
</assembly-descriptor>
```

<security-role>元素中只有惟一的元素<role-name>



# 定义角色…… 更好的办法

## 带描述的<security-role>元素

描述（<description>）是<security-role>元素中的可选元素。虽然可以没有这个元素，不过请考虑一下。




想想那些可怜的部署人员吧，不仅工作压力大，报酬还低（实际上，你可能就是部署人员，因为你很有可能身兼三职：bean提供者、应用组装人员和部署人员，有时候甚至还要担负系统管理员的职责）。

**他又不是什么都知道。**

有了描述，部署人员就能更容易地把自造的角色映射到公司（要运行这个应用）中的实际用户组。

```
<assembly-descriptor>

   <security-role>
      <description>
          This role is for the employees who have the
          power to view and change all employee payroll
          information.
      </description>
      <role-name>Payroll Director</role-name>
   </security-role>

   <security-role>
      <description>
          This role is for the employees who have the
          power to view all employee information, and
          change some pieces
      </description>
      <role-name>Payroll Admin</role-name>
   </security-role>

   <security-role>
      <description>
          This role is for the employees who have the
          power to view all employee payroll
          information.
      </description>
      <role-name>Payroll Assistant</role-name>
   </security-role>
...
</assembly-descriptor>
```



# 定义方法许可

## <method-permission>元素

bean的home接口和组件接口中的所有方法都有可能由客户调用。既然应用组装人员已经定义了角色，接下来可以定义每个角色能够调用哪些方法。就像定义安全角色一样，要把方法许可放在部署描述文件的**<assembly-descriptor>**部分。

```
<assembly-descriptor>

   <method-permission>
        <role-name>Payroll Director</role-name>

        <method>
           <ejb-name>ChangePay</ejb-name>
           <method-name>*</method-name>
        </method>

        <method>
           <ejb-name>Employee</ejb-name>
           <method-name>*</method-name>
        </method>

   </method-permission>


   <method-permission>
        <role-name>Payroll Admin</role-name>

        <method>
           <ejb-name>ChangePay</ejb-name>
           <method-name>reimburse</method-name>
        </method>

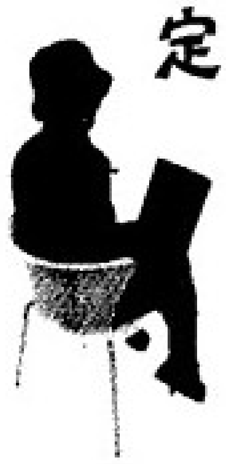
        <method>
           <ejb-name>ChangePay</ejb-name>
           <method-name>giveBonus</method-name>
        </method>

        <method>
           <ejb-name>Employee</ejb-name>
           <method-name>getSalary</method-name>
        </method>
     ...
   </method-permission>
  ...
</assembly-descriptor>
```

定义方法许可时，首先是一个角色，然后指出这个角色允许调用这个bean中的哪些方法。

这个方法许可指出，"角色为'Payroll Director'的客户可以调用ChangePay bean客户视图（即home和组件接口）中的所有方法以及Employee bean客户视图中的所有方法"

这个方法许可指出"角色为'Payroll Admin'的客户可以调用ChangePay bean的reimburse和giveBonus方法，以及Employee bean的setSalary方法"。



# 定义方法许可

## 指定方法有3种不同方式

前面已经看到了指定一个bean方法名的两种做法：使用通配符（*），这表示bean中的所有方法；或者使用方法的实际名。不过只有名字往往是不够的。这在前面曾经谈到过，关于事务的一章中在指定事务属性时曾经遇到过同样的问题。如果有重载方法会怎么样呢？

你的设计中很可能将一个重载方法的所有版本都同样处理。但是如果有其他想法，比如希望一个特定的安全角色只能访问一个重载方法的某一个版本，而不能调用其他的版本，此时就可以使用一个可选的**<method-params>**元素。

**① 使用通配符(*)表示所有方法。**

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-name>*</method-name>
</method>
```

星号(*)是通配符，表示 bean接口中的所有方法。

**② 只使用一个方法名，表示同名的所有方法，不论参数如何，也不管是在home接口中还是组件接口中。**

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-name>takeOver</method-name>
</method>
```

这表示名为"takeOver"的所有重载方法都可访问。

**③ 通过方法名和参数，可以区别重载方法。**

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-name>takeOver</method-name>
    <method-params>
        <method-param>String</method-param>
    </method-params>
</method>
```

这里只指定了取一个String参数的takeOver方法，不包括其他重载版本。

# 方法许可通过取并集相结合!

方法许可相互之间的关系与事务属性不同。对于事务属性，使用一个通配符就意味着，“这个bean中的所有方法都有这个属性，除非另外说明，也就是除非在一个不同的容器托管事务标记中指定一个特定方法”。换句话说，如果指定一个特定的方法，会覆盖通配设置。

但是对于方法许可，使用通配符是说，“这个bean中的所有方法都可以由这个角色访问”。不管在其他方法许可中做了什么设置，都不会改变这一点。

## 事务属性

```
<container-transaction>
   <method>
      <ejb-name>BigBean</ejb-name>
      <method-name> * </method-name>
   </method>
   <trans-attribute>RequiresNew</trans-attribute>
</container-transaction>

<container-transaction>
   <method>
      <ejb-name>BigBean</ejb-name>
      <method-name>useOldDatabase</method-name>
   </method>
   <trans-attribute>NotSupported</trans-attribute>
</container-transaction>
```

这说明，BigBean中的所有方法要使用RequiresNew属性，但useOldDatabase方法除外，它要使用NotSupported属性。

第二个<container-transaction>覆盖了前面的通配设置，useOldDatabase方法从使用RequiresNew属性的方法列表中取出，把它移到了使用NotSupported属性的列表中。

## 方法许可

```
<method-permission>
   <role-name>Minion</role-name>

      <method>
        <ejb-name>WorldDomination</ejb-name>
        <method-name>*</method-name>
      </method>
</method-permission>


<method-permission>
   <role-name>Boss</role-name>

      <method>
        <ejb-name>WorldDomination</ejb-name>
        <method-name>learnPlan</method-name>
      </method>
</method-permission>
```

这说明，Minion角色可以访问WorldDomination的所有方法。

而且Boss角色只能访问WorldDomination的learnPlan方法。

第二个<method-permission>会增加到第一个<method-permission>。

## 当心<unchecked/>

可以把<unchecked/>认为是方法许可中取代<role-name>的元素。但是由于方法许可采用并集的方式结合，所以如果使用<unchecked/>为一个方法定义了方法许可，那么你为该方法建立的其他方法许可就没有什么意义了。如果只有一个小<unchecked/>，这意味着这个方法谁都能调用，而不论方法的主体或安全角色是什么！

### 使用<unchecked/>定义的方法许可

```
<method-permission>

   <role-name>AccountManager</role-name>

      <method>
        <ejb-name>Account</ejb-name>
        <method-name>increaseLimit</method-name>
      </method>
</method-permission
```

这说明AccountManager可以调用Account bean上的increaseLimit()方法。

```
<method-permission>

   <unchecked/>

      <method>
        <ejb-name>Account</ejb-name>
        <method-name>*</method-name>
      </method>
</method-permission
```

<unchecked/>取代了<role-name>元素

谁会去管前面的方法许可？？！！这里又有Account的一个方法许可，称所有方法都不会检查，所以无论关于Account bean方法再说什么都没什么影响。不管怎样，这个bean的所有方法都是非受查的（都可以访问）！

**<unchecked/>元素会覆盖一个方法的所有其他方法许可。**




**如果一个bean有两个同名的方法，但是一个方法在home接口中，另一个方法放在组件接口中，这样一个bean该怎么处理呢？**

（不错，我们前面说的有点错误，实际上描述一个方法有4种方法：通过通配符（*）、通过方法名、通过方法名和参数，以及通过方法名和接口）

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>takeOver</method-name>
</method>
```

<method-intf>的值必须是以下4个值之一：

```
<method-intf>Remote</method-intf>

<method-intf>Home</method-intf>

<method-intf>Local</method-intf>

<method-intf>LocalHome</method-intf>
```

# 部署人员的任务：把实际的人映射到抽象的角色



应用组装人员很了解应用，但是部署人员才了解操作环境。我们（以及规范）都用了“操作环境”这个词，这实际上就是指运行应用的企业。也许公司购买的是成品应用，也可能是他们自己构建的应用。这都没关系。有部署人员在就行，他很清楚这个环境。最重要的是，他知道公司里是如何管理安全的（例如，公司可能有所有员工的名字和口令，并把这些信息作为LDAP系统的一部分）。要论谁知道应用组装人员所建立的抽象角色应当如何映射到公司中实际的人和工作组，非部署人员莫属。

部署人员有两项主要任务：

## ① 为应用指定安全领域和主体域。

- 运行应用的公司有实际的员工。这些员工要采用某种方式向服务器完成认证，可能就是提供一个名字和一个口令。操作环境中的安全信息必须以某种方式配置到服务器中，这样服务器可以分清实际上是谁调用方法。这些并不属于EJB规范的范畴！换句话说，这是特定于开发商的。

## ② 把用户和/或用户组映射到抽象安全角色。

- 应用组装人员建立了最适合这个工资应用的抽象安全角色。但是这些角色在要运行应用的公司中并不表示什么。部署人员必须建立Annie（应用组装人员）所定义的角色［Payroll Director（主任）、Payroll Admin（管理员）、Payroll Assistant（助手）］与部署人员所在公司中实际的人［Payroll Manager（经理）、Payroll Supervisor（主管）、HR Admin（人力资源管理员）］之间的映射。

这同样不在EJB规范范围内，是特定于开发商的。

注意：规范使用了“逻辑安全角色”一词，这指的就是我们所说的“抽象安全角色”，不过我们更倾向于“抽象”这个词。只要知道“逻辑安全角色”与“抽象安全角色”含义相同（指的是同一个东西）就行了。

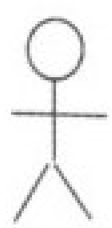
**Relax** **考试时，你不用知道部署人员怎么完成他的任务，只要知道他必须做这些工作就行了。**

要记住，考试并不要求你了解任何特定于开发商的功能。但是你确实需要知道，要完成实际的人与应用组装人员建立的抽象角色之间的映射，这是部署人员的职责。

# 主体和角色，用户和用户组

Dan Johnsson向系统完成了认证（使用他的登录名和口令），现在表示为Principal类型的一个实例。采用一种特定于开发商的方式，部署人员告诉EJB应用，Dan的主体（Principal）与一个或多个角色相关联。所以，只要Dan作为一个客户调用bean客户接口上的某个方法时，他的Principal（以及指定给这个Principal的角色）就会随调用传播。这说明，（概念）调用栈上的每个调用都会把Dan的Principal看作是调用者。

## EJB中



系统中的抽象演员（通常映射到一个人）

### ① 主体（Principal）

这是得到系统认证的客户，表示为一个java.security.Principal对象。这个主体（Principal）是与一个名关联的某个东西的抽象表示，但是不能保证主体名一定与客户的登录名一致。这取决于你的系统如何处理安全认证。而且不要想当然地认为主体就是一个单一的个体。有时主体表示的是更大的用户组，如SysAdmin（系统管理员）。主体与应用组装人员定义的一个或多个抽象安全角色相关联。bean对Principal对象只能做一件有用的事情，这就是得到它的名字（aPrincipal.getName()），但是这很危险，因为除非你知道bean在哪个环境中运行，否则无法确切地知道这个名字表示的是什么。真是烦人。

*Manager*
*KeyAccount*
*Customer*
*PayrollDirector*
*SwedishTeam*
*SwingDancers*

### ② 角色（Role）

EJB中，安全角色是应用组装人员定义的一个抽象角色，尽管它并不与实际世界的任何东西匹配，但是部署人员会把公司（运行这个应用的公司）中实际的用户和用户组映射到这些抽象角色。例如，Employee（员工）抽象角色可以映射到部署人员所在公司的“Slaves”组。

## 操作环境中



### ③ 用户（User）

用户是实际环境中的人。这是实际的用户，活着的、能呼吸的人。在实际的环境中会以某种方式来表示这些人，但是在EJB中，我们当然无法提前知道具体的情况，比如公司可能使用一个LDAP服务器来维护用户的安全信息。一般地，用户会映射为一个登录名和口令，这个信息会存储在某个地方，如果你幸运，选择了合适的服务器，你的EJB服务器可以直接配置为保存安全信息的系统。如果EJB服务器与你公司用来存储用户安全信息的机制不兼容，你就必须老老实实地坐在那里，在EJB服务器的性质文件中键入10000个用户名和口令，这肯定是你不希望的。

*Customer*
*PayrollManager*
*Sweden*
*SalsaDancers*

### ④ 用户组（Group）

一般地，安全信息会把用户组织为一个或多个组，通常按部门或位置分组。部署人员可以直接从用户组映射到抽象安全角色，不过他可能还必须映射单个的用户。

OK，我知道写代码的时候不用考虑这个……但是有个问题困扰着我。请看部署描述文件，其中你可以说谁能访问哪种bean类型，但是你不能说可以访问那种bean类型的哪个实例。



我很理解他的想法，因为我也考虑过同样的问题。如果我希望员工能且仅能访问他们自己的一些数据，该怎么办呢？怎么才能建立单个实体级的控制？否则，我就必须阻止所有员工访问这些方法，或者让所有员工都能访问可以处理任何实体bean的方法。比如，我怎么才能让employee调用getSalary()，但只能看到他自己的工资呢？



# 类级安全与实例级安全

**当声明方式实现安全不够用时，你可能需要程序方式实现安全来限制一个特定bean实例的访问。**

到目前为止，我们看到的所有安全都是声明方式的……没有硬编码写到bean代码中。声明方式实现安全很棒，因为它支持基于组件的开发，你可以在部署时定制bean，而无需去动代码。公司A可能在一个应用中使用一个bean，它需要某种访问控制，而这与同样使用这个bean的公司B采用的访问控制方法完全不同。或者甚至同一个公司将同一个bean用做两种用途时也可能需要不同的访问控制。

但是声明方式的安全是在类级实现的。你只能指定一个特定角色可以调用哪些方法，但是这说明，这个角色可以调用该bean类任何实例的方法。如果需要实例级的安全，在部署描述文件中是无法做到的。不过你可以采用程序方式实现安全，你应该对这个不陌生……前面已经看到过SessionContext和EntityContext中的两个与安全有关的方法。




```
public void doSecurity() {
    java.security.Principal p = context.getCallerPrincipal();
    String name = p.getName();
    // 这里做一个比较，
    // 相对于应当与主体名匹配的bean中的
    // 持久字段，检查name
}
```

不过要小心！不能保证对主体调用getName()方法得到的名字与用户的登录名一定匹配。所以只要在bean中采用了程序方式实现安全，就必须做好心理准备，bean的可重用性和可移植性会严重降低。而且，这说明bean提供者可能需要与操作环境更紧密地耦合，这通常不是你希望的。因为bean提供者有一个假设，就是从getName()方法得到的名字会与实体bean中的一个持久字段匹配，而这在所有环境中可能都不成立，这取决于你的系统如何管理安全。

# 使用程序方式实现安全来定制方法

前面已经看到了getCallerPrincipal()，通过这个方法可以知道容器认为到底是谁在调用。不仅如此，我们还可以使用isCallerInRole()方法来检查主体（不管它是谁，在这种情况下我们并不关心主体是谁）是不是某个特定的角色（这个角色才是我们关心的）。假设你有一个私有的定价方法，由某个业务方法调用，它要检查这个顾客是否认证为特殊VIP顾客组中一员。如果他确实是VIP顾客，就可以做特殊处理。也许会给他一个折扣，也许会对他提价。这要看你的业务逻辑怎么说了。

```
private void determinePrice() {

    if (context.isCallerInRole( "VIPCustomer" )) {
        // 善待这个顾客
    } else {
        // 不那么殷勤地对待这个顾客
    }
}
```

把安全信息硬编码写到程序中有一些很明显的问题（禁止重用、无法以可移植的方式工作的可能性更大，等等），除此以外，还有一个严重的问题，bean提供者到底怎样才能提前知道应用组装人员要指定哪些角色？

bean提供者并不知道。或者，至少不应该知道。我们很清楚。在实际中，bean提供者和应用组装人员如果不是同一个人，也往往有很密切的关系（当然，这是从专业的角度来讲的）。不过，这里我们还是要力争得到一个基于组件的开发模型，至少我们希望应用组装人员拿到bean提供者提供的bean（不管是什么样的bean），都能很好地用于她自己的应用，而这意味着要把这些bean与从其他提供者那里得到的bean集成。

所以现在假设有这样一种情况：Annie（应用组装人员）利用4个现有的bean构建一个新应用，其中两个bean来自不同的提供者。这些提供者并不为同一家公司工作，而且相互之间不认识。他们怎么可能提前交流，怎么可能知道Annie将为她的应用选择什么角色呢？

**利用isCallerInRole(),可以使用安全信息来定制方法运行的方式，这取决于哪个角色调用这个方法。**

**这通常不是处理安全的好办法。要控制对一个方法的访问，最好还是使用方法许可来禁止未授权的角色调用方法。**

**不过，还是可以使用调用者的角色信息在代码中做其他一些与安全无关的事情。**

# isCallerInRole()的问题……

## bean提供者硬编码了一个角色名，但是他怎么知道应用组装人员要使用什么角色呢？如果有不只一个bean提供者该怎么办？

我要在代码中把大客户的角色叫做“VIPCustomer”。



Bill
（bean提供者A）

发现问题了吗？我要把这些bean集成到一个应用中，但是Bill在bean中硬编码写入的名字是“VIPCustomer”，而RaySunshine认为这个角色应该叫“EnlightenedCustomers”。所以我必须寻找一种方法把他们这些完全自造的名字映射到我在部署描述文件中建立的抽象安全角色。我要告诉应用，他们的这两个名字都表示“KeyAccounts”。另外，如果我没有看他们的源代码，怎么知道他们使用了这些名字呢？？？

我会把重要客户的角色叫做“EnlightenedCustomers”。



RaySunshine
（bean提供者B）



Annie需要在她的应用中使用这两个bean，但是她想把角色名定义为“KeyAccounts”

应用组装人员必须在他希望为这个应用定义的抽象角色与程序员硬编码的（完全自造的）角色名之间建立映射。

抽象角色名：KeyAccounts

bean A的角色名：VIPCustomer

bean B的角色名：EnlightenedCustomers

同一个角色有3个不同的名字！Annie希望部署人员将实际的顾客组映射到“KeyAccounts”，而不是另外两个名字，不过这两个名字硬编码在代码中，而她无法去改Java源代码！

## 把声明方式的安全角色映射到程序员硬编码的（自造）角色

应用组装人员使用<role-link>元素在安全角色引用（程序员自造的硬编码角色名）和安全角色（应用组装人员为这个应用选择的抽象角色名）之间建立映射。

---

### bean提供者




bean提供者必须告诉应用组装人员，他在代码中做了什么，为此要放入一个<security-role-ref>标记

```
<enterprise-beans>
   <session>
      <ejb-name>ShoppingCart</ejb-name>
      ...
      <security-role-ref>
         <description>
            this role should be assigned to the
            accounts that get special VIP pricing
         </description>
         <role-name>VIPCustomer</role-name>
      </security-role-ref>
     ...
   </session>
</enterprise-beans>
```

这些名字必须匹配。你放的不是"真正的"角色名，而是用作为isCallerInRole()参数的名字。

```
context.isCallerInRole("VIPCustomer")
```

---

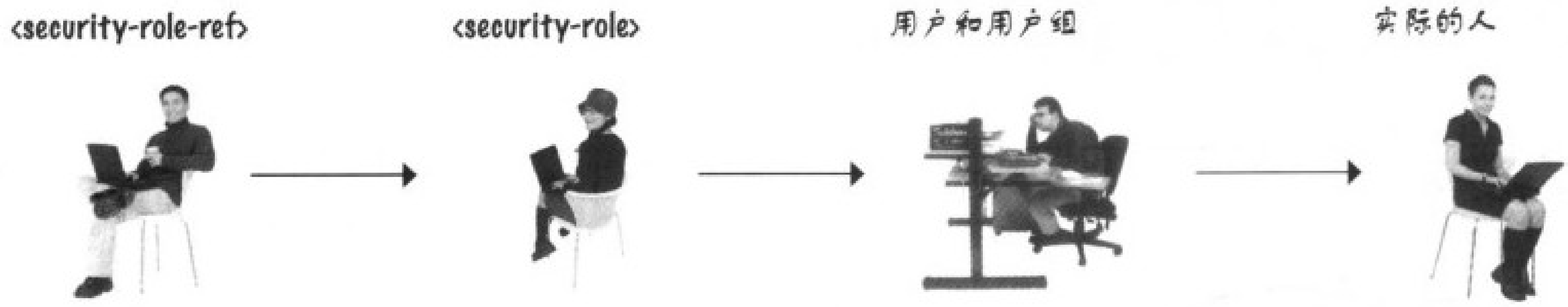
### 应用组装人员




对于bean提供者在<security-role-ref>中放的名字，应用组装人员要相应地增加一个<role-link>，从而把硬编码的角色映射到应用组装人员的真正的抽象角色。

```
<security-role-ref>
   <description>
      this role should be assigned to the
      accounts that get special VIP pricing
   </description>
   <role-name>VIPCustomers</role-name>
   <role-link>KeyAccounts</role-link>
</security-role-ref>
```

## 完整的安全映射图

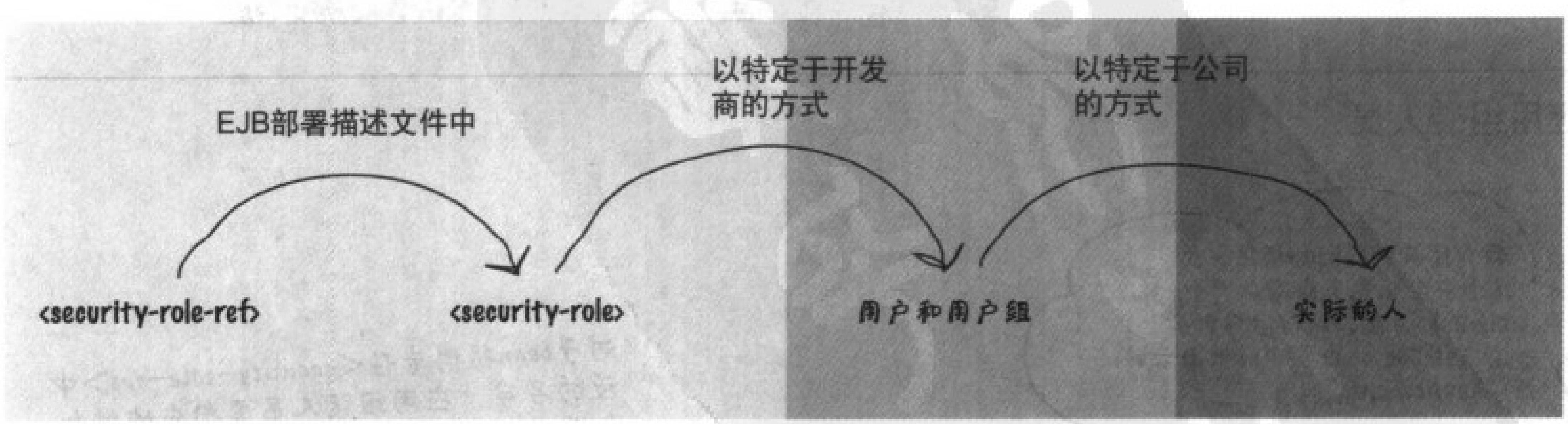



应用组装人员使用<role-link>元素把程序员硬编码的安全角色引用映射为她建立的抽象安全角色。

部署人员使用一个特定于开发商的映射（可能是一个性质文件），将应用组装人员的抽象安全角色映射到部署人员所在公司的用户和用户组。

操作环境中的某个人建立实际公司的用户和用户组。这可能是某个负责为员工指定登录信息和口令的系统管理员。

### 映射在哪里发生，如何发生




这是规范中的部分，也是考试中要求你知道的部分

这些要从你的开发商/产品文档中了解

这不是bean开发人员的任务

# 使用<run-as>安全身份假装成别人来调用……

一个客户调用一个方法时，容器总是知道这个客户的主体，这包括部署人员分配给这个主体的抽象安全角色。而且要记住，在客户原来的方法完成工作的过程中，调用者的安全上下文会通过应用传播。概念调用栈中每一个被调用的方法在得到调用的同时都会得到这个安全上下文。

但是…… 假设你不希望客户的安全上下文传播，怎么办？假设客户调用bean A，而bean A接下来调用了bean B，你希望bean B认为是别人在调用。换句话说，如果你希望bean A假装成别人，该怎么做？这样的话，bean A调用的所有bean都会认为调用者的主体（和角色）不是原来的客户，而是别人。

为什么要这样做？bean B可能有更严格的访问控制。也许bean B不允许外面的客户调用它的方法，所以对于映射到客户用户和用户组的任何抽象角色，没有建立相应的方法许可。但是，你可能有一个刚为其他bean建立的特殊角色，只要是这种角色的bean，bean B就可以接收这些bean的调用。




**如果你希望bean假装成别人，而不是原来调用方法的客户**

```
<enterprise-beans>
  <session>
      <ejb-name>BeanA</ejb-name>
      ...
      <security-identity>
         <run-as>
            <role-name>SuperBean</role-name>
         </run-as>
      </security-identity>
      ...
  </session>
</enterprise-beans>
```

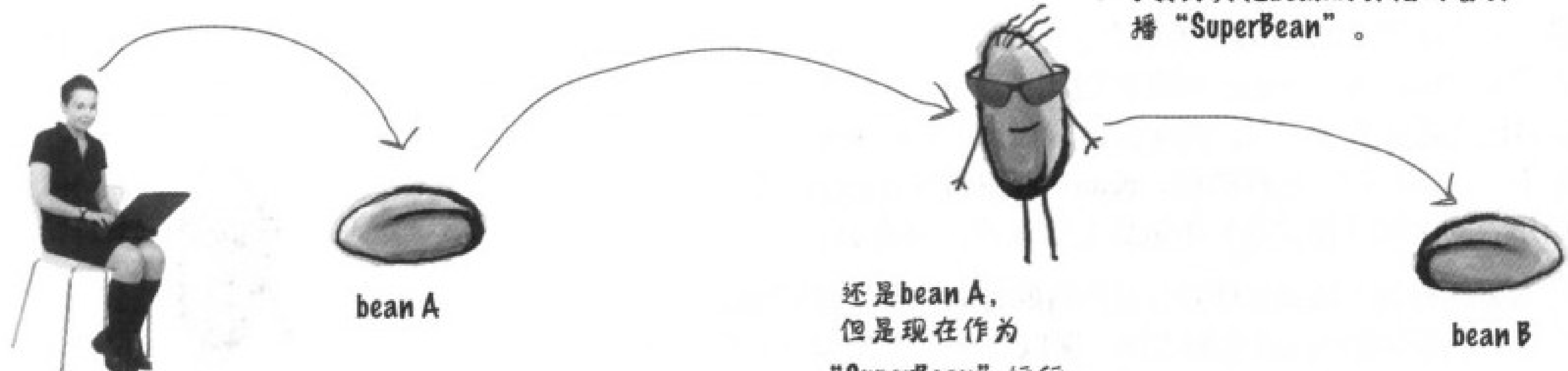
role-name必须与应用组装人员建立的一个实际抽象安全角色匹配。

**明确地指出你希望使用原来调用客户的身分**

```
<enterprise-beans>
   <session>
      <ejb-name>BeanA</ejb-name>
      ...
      <security-identity>
         <use-caller-identity/>
      </security-identity>
      ...
   </session>
</enterprise-beans>
```

如果根本没有放<security-identity>元素，就会得到这种默认情况。所以不用明确指出，不过这么做也是可以的。

# 采用<run-as>的安全上下文传播




**bean A的<security-identity>并不是bean A看到的调用者，而是当bean A调用bean B时bean B所看到的身份。**

**换句话说，<security-identity>不会改变到来客户的身份，只会改变bean在调用其他bean时传播的传出身份。**

**消息驱动bean不能用<use-caller-identity/>!!**

会是哪个调用者？消息驱动bean根本没有调用客户！所以要是部署描述文件中显示一个消息驱动bean有一个<use-caller-identity/>安全身份，一定要当心。不过，消息驱动bean可以有一个<run-as>身份。而且，以前说过，别忘了消息驱动bean不能向它的MessageDrivenContext要调用客户的安全身份。<security-identity>并不是指这个bean认为谁在调用它，而是这个bean在调用其他bean时假装成什么人。

（就像你用“Jennifer Lopez”的名字在旅店登记一样，尽管你实际上是“Simon Roberts”……但是旅店的人并不清楚。对他来说，你就是J-Lo。）

# COFFEE CRAM

## 模拟测验

**1** 关于EJB的安全，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 所有安全策略都必须用声明方式表述
- ❑ B. 完成一个方法调用时，所处的默认安全身份就是组件的创建者
- ❑ C. 通过使用EJB，可以在部署描述文件中使用EJB QL声明方法许可
- ❑ D. 可以在方法级建立安全授权
- ❑ E. 可以在实例级建立安全授权

**2** 对于无需授权检查就能运行的方法，下面哪种说法是正确的？（选出所有正确的答案。）

- ❑ A. 可以列在<exclude-list>元素中
- ❑ B. 可以列在<unchecked>元素中
- ❑ C. 使用<unchecked>元素时，应当放在部署描述文件中通常放<role-name>元素的地方
- ❑ D. 一个方法许可关系同时指定了<unchecked>和一个安全角色时，容器会使角色

**3** 通常应当由哪些人为应用定义适当的安全策略？（选出所有正确的答案。）

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 系统管理员
- ❑ E. 服务器提供者

**4** 一个有状态会话bean可以在以下哪个方法中调用**isCallerInRole**方法来完成一个安全检查？（选出所有正确的答案。）

- ❑ A. **ejbCreate()**
- ❑ B. **ejbActivate()**
- ❑ C. **ejbPassivate()**
- ❑ D. 以上都不行

**5** 要保证一个应用安全，bean提供者的职责是什么？（选出所有正确的答案。）

- ❑ A. 为应用指定安全领域
- ❑ B. 完成主体的认证
- ❑ C. 把客户所用的主体映射到为bean定义的主体
- ❑ D. 以上都不是

**6** 一个CMP实体bean可以在以下哪个方法中调用getCallerPrincipal()方法来完成一个安全检查？（选出所有正确的答案。）

- ❑ A. **ejbCreate()**
- ❑ B. **ejbActivate()**
- ❑ C. **ejbPassivate()**
- ❑ D. **ejbPostCreate()**
- ❑ E. 业务方法

**7** 给定以下方法，一个BMP实体bean可以在以下哪个方法中调用**getCallerPrincipal()**方法来完成一个安全检查？（选出所有正确的答案。）

- ❑ A. **ejbLoad()**
- ❑ B. **ejbStore()**
- ❑ C. **ejbPassivate()**
- ❑ D. **ejbPostCreate()**
- ❑ E. 业务方法

**8** 关于部署描述文件中一个特定bean的方法许可，下面哪种说法是正确的？（选出所有正确的答案。）

- ❑ A. 通配符可以指示bean的所有方法
- ❑ B. 各个重载方法无法相互区别
- ❑ C. home接口中的一个方法无法与组件接口中的一个同名方法相区别
- ❑ D. 可以指示单个的方法

**9** **<role-name>**元素可以用在哪个与安全相关的部署描述文件元素中？（选出所有正确的答案。）

- ❑ A. **<security-role>**
- ❑ B. **<run-as>**
- ❑ C. **<method-name>**
- ❑ D. **<exclude-list>**
- ❑ E. **<security-role-ref>**

**10** 实现EJB应用的安全时，关于每个人的职责下面哪种说法是正确的？（选出所有正确的答案。）

- ❑ A. 应用组装人员通常要指定应用中什么时候应当使用run-as身分
- ❑ B. bean提供者将安全角色引用映射为安全角色
- ❑ C. bean提供者通常负责把安全领域和主体域映射到应用
- ❑ D. 部署人员把安全角色引用映射为安全角色

**11** 在**<security-role-ref>**部署描述文件元素中，哪些子元素是可选的？（选出所有正确的答案。）

- ❑ A. **<role-name>**
- ❑ B. **<role-link>**
- ❑ C. **<description>**
- ❑ D. 以上都不是可选的

# COFFEE CRAM
# 模拟测验答案

**1** 关于EJB的安全，哪些说法是正确的？（选出所有正确的答案。） （规范:434～435）

❑ A. 所有安全策略都必须用声明方式表述

❑ B. 完成一个方法调用时，所处的默认安全身份就是组件的创建者

❑ C. 通过使用EJB，可以在部署描述文件中使用EJB QL声明方法许可

✔ D. 可以在方法级建立安全授权

❑ E. 可以在实例级建立安全授权

**2** 对于无需授权检查就能运行的方法，下面哪种说法是正确的？（选出所有正确的答案。） （规范:443）

❑ A. 可以列在<exclude-list>元素中 — 不能被调用的方法才放在这里

✔ B. 可以列在<unchecked>元素中

✔ C. 使用<unchecked>元素时，应当放在部署描述文件中通常放<role-name>元素的地方

❑ D. 一个方法许可关系同时指定了<unchecked>和一个安全角色时，容器会使角色

**3** 通常应当由哪些人为应用定义适当的安全策略？（选出所有正确的答案。） （规范:435～436）

❑ A. bean提供者

✔ B. 应用组装人员

✔ C. 部署人员

❑ D. 系统管理员

❑ E. 服务器提供者

**4** 一个有状态会话bean可以在以下哪个方法中调用`isCallerInRole`方法来完成一个安全检查？（选出所有正确的答案。） (规范：80)

- [x] A. `ejbCreate()`
- [x] B. `ejbActivate()`
- [x] C. `ejbPassivate()`
- [ ] D. 以上都不行

(规范：448～454)

**5** 要保证一个应用安全，bean提供者的职责是什么？（选出所有正确的答案。）

- [ ] A. 为应用指定安全领域　　这通常是部署人员的任务
- [ ] B. 完成主体的认证
- [ ] C. 把客户所用的主体映射到为bean定义的主体
- [x] D. 以上都不是

(规范：179～180)

**6** 一个CMP实体bean可以在以下哪个方法中调用getCallerPrincipal()方法来完成一个安全检查？（选出所有正确的答案。）

- [x] A. `ejbCreate()`
- [ ] B. `ejbActivate()` — 谁是客户？
- [ ] C. `ejbPassivate()`
- [x] D. `ejbPostCreate()`
- [x] E. 业务方法

**7** 给定以下方法，一个BMP实体bean可以在以下哪个方法中调用`getCallerPrincipal()`方法来完成一个安全检查？（选出所有正确的答案。） (规范：257)

- [x] A. `ejbLoad()`
- [x] B. `ejbStore()`
- [ ] C. `ejbPassivate()` — 谁是客户？
- [x] D. `ejbPostCreate()`
- [x] E. 业务方法

**8** 关于部署描述文件中一个特定bean的方法许可，下面哪种说法是正确的？（选出所有正确的答案。） （规范：444）

- ☑ A. 通配符可以指示bean的所有方法
- ❑ B. 各个重载方法无法相互区别 这可以用<method-intf>元素来区别
- ❑ C. home接口中的一个方法无法与组件接口中的一个同名方法相区别
- ☑ D. 可以指示单个的方法

**9** `<role-name>`元素可以用在哪个与安全相关的部署描述文件元素中？（选出所有正确的答案。） （规范：444～447）

- ☑ A. `<security-role>`
- ☑ B. `<run-as>`
- ❑ C. `<method-name>`
- ❑ D. `<exclude-list>`
- ☑ E. `<security-role-ref>`

**10** 实现EJB应用的安全时，关于每个人的职责下面哪种说法是正确的？（选出所有正确的答案。） （规范：446～449）

- ☑ A. 应用组装人员通常要指定应用中什么时候应当使用run-as身份
- ❑ B. bean提供者将安全角色引用映射为安全角色 通常由应用组装人员定义
- ❑ C. bean提供者通常负责把安全领域和主体域映射到应用
- ❑ D. 部署人员把安全角色引用映射为安全角色

**11** 在`<security-role-ref>`部署描述文件元素中，哪些子元素是可选的？（选出所有正确的答案。） （规范：527）

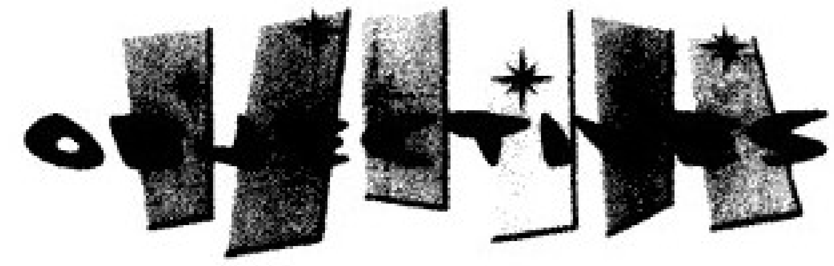
- ❑ A. `<role-name>`
- ☑ B. `<role-link>` 通常由应用组装人员定义
- ☑ C. `<description>`
- ❑ D. 以上都不是可选的

# 12 bean的环境

# 部署的快乐




**bean简直让你费尽力气。**你要编写代码，完成编译，进行测试。这个过程反反复复地做了无数次。你最不希望的就是，仅仅因为部署配置中的一个小小的改变，就要对已经测试过的源代码再做改动。也许是数据库的名字变了，也可能是在程序中硬编码写入了一个税率（记住，bean不能访问性质文件）。另外，如果你根本拿不到源代码怎么办？这些bean可能是你从Beans-r-Us得到的，而他们没有卖给你源代码（或者你根本没有这个预算）。EJB通过部署描述文件和bean的一个特殊环境可以支持bean重用。



## bean的环境

### 标准大纲：

13.1 关于企业bean的JNDI命名，明确哪些说法或示例是正确的，哪些不正确。

13.2 部署描述文件中有对应环境项、EJB引用和资源管理器连接工厂引用的元素，关于这些元素的用途和/或使用，明确哪些说法或示例是正确的，哪些不正确，包括给你一个代码清单，关于一个特殊的部署描述文件元素，要求你查看这段代码是否合适、是否正确。

13.3 给定一组职责，明确其中分别哪些是部署人员、bean提供者、应用组装人员、容器、系统管理员或某种组合的职责。

1.2 给定一组技术规范，明确哪些是EJB 2.0容器所必需的。

1.3 关于EJB编程限制，明确哪些说法或示例是正确的，哪些不正确。

1.4 将EJB角色与角色职责的有关描述加以匹配，描述中可能包括部署描述文件信息。

1.5 给定一个列表，明确哪些是对ejb-jar文件的要求。

### 具体含义：

必须知道可以采用何种方式在部署时定制一个bean，而无需修改源代码。bean使用自己的特殊环境完成JNDI查找时，你必须知道代码如何与部署描述文件中的内容相关联，还要了解许多可谓吹毛求疵的细节问题，往往正是这些细节决定了部署是否成功。如果没有注意这些问题，可能无法部署。或者更糟糕的……可能可以部署，但是在运行一段时间之后突然崩溃。

你要知道哪些方面是EJB 2.0规范可以保证的，而哪些方面不能保证。例如，EJB 2.0容器不必支持JXTA、JMX、Servlets或JSP，但是必须支持JNDI、JMS和JAXP。你要知道EJB到底能给你什么。

你还要知道，在EJB中，如果想建立一个可以在任何EJB 2.0兼容的容器上移植的bean，哪些可以做，哪些不能做。例如，不能监听一个ServerSocket，但是可以创建一个客户Socket。不能访问服务器的本地文件系统，也不能建立你自己的线程。但是让一个bean类扩展另一个bean类是允许的。

你要了解EJB角色（bean提供者、应用组装人员等等），还要知道在开发和部署期间谁做什么。尽管这方面的问题不会很多，但是这一部分对于参加考试的人来说有一定的难度，所以如果我是你，就会很仔细地学习这一部分。

最后一点，你要知道一个ejb-jar文件中哪些应该有，哪些不应该有。例如，必须有home接口和组件接口（除非使用的是一个消息驱动bean），但其中不能有桩。




**你喜欢每次需要改变bean行为之时都修改你的源代码吗？是的话，请举手。**

当然，如果只是有人改变了数据库的名字，你肯定不想去动你的源代码。你知道，通过修改部署描述文件就完全可以改变事务属性和安全访问。不过，请等等……还不只如此！

每个EJB容器都会为你的bean提供它自己的一个特殊环境，你的bean要使用这个特殊环境来查找4样东西：

- **环境项**

  环境项是一些部署时值（这样你就不用在代码里放诸如当前折扣率或税率之类的值了）。

- **资源管理器连接工厂**

  你要使用这些资源管理器连接工厂来得到一个数据库连接。

- **企业bean引用**

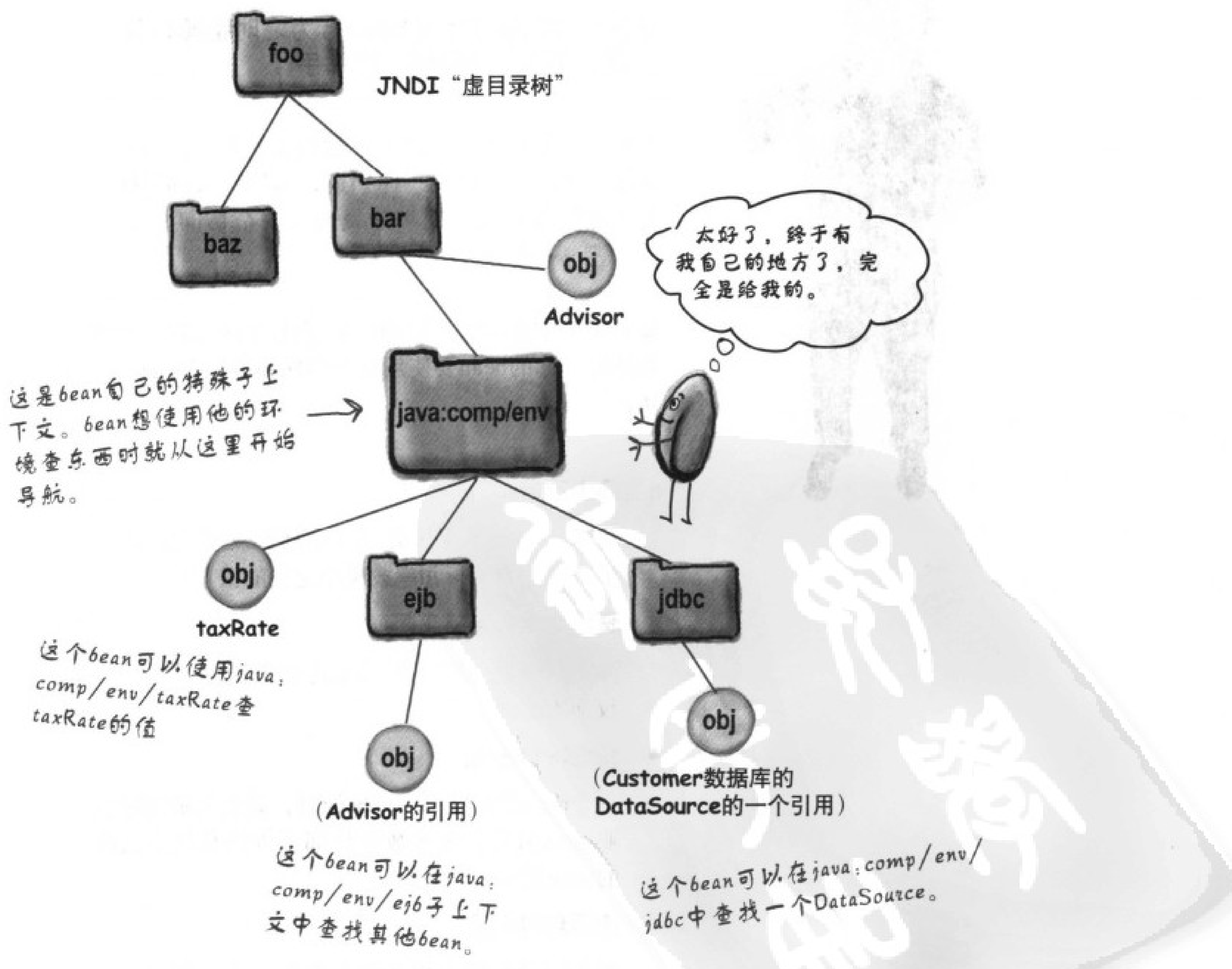
  一个bean想查找另一个bean时，就会大量使用企业bean引用！大多数设计都至少涉及某个层次的bean到bean通信。

- **资源环境引用**

  要用资源环境引用从服务器得到某个所谓“受管对象”的引用，如JMS目标。

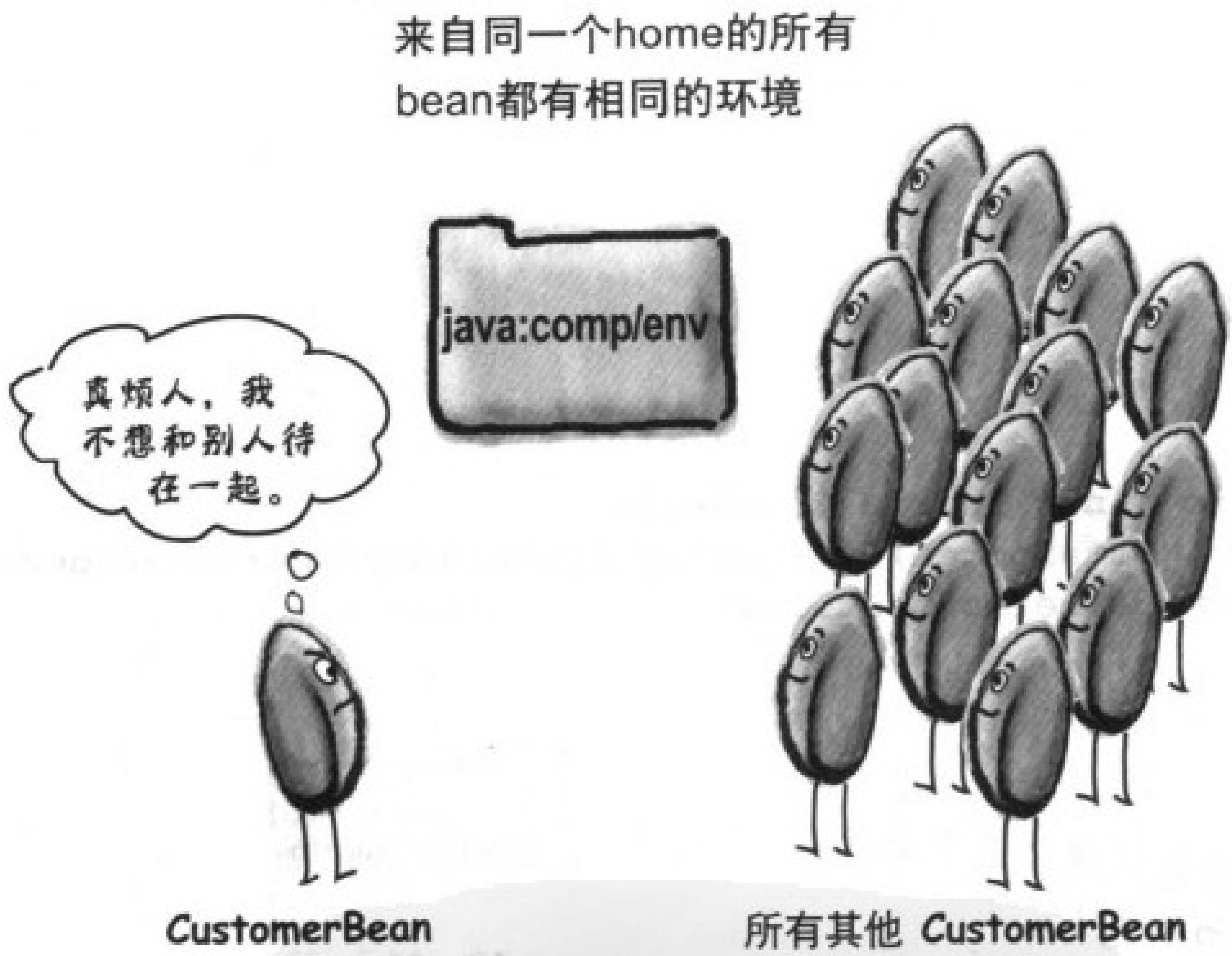
# bean的特殊位置java:comp/env

bean会有它自己的一个JNDI上下文。这是一个专为这个bean提供的特殊位置，bean可以在这里进行查找。上下文总是从InitialContext开始，但是bean想查东西时往往从java:comp/env子上下文开始导航。



# 但不是每个bean实例都有自己的环境……而是每个bean home有一个环境

同一个home的所有bean都有共同的bean环境。如果把某个特定的bean类型在服务器上部署了3次，3次部署都会有不同的home，所以一共有3个不同的环境。




```
InitialContext ic = new InitialContext();
Double discount =  (Double) ic.lookup("java:comp/env/custDiscount");
```

不论是哪个CustomerBean实例运行这个代码，这个值都是一样的。

# 环境子上下文

java:comp/env/jdbc/CustomerDB

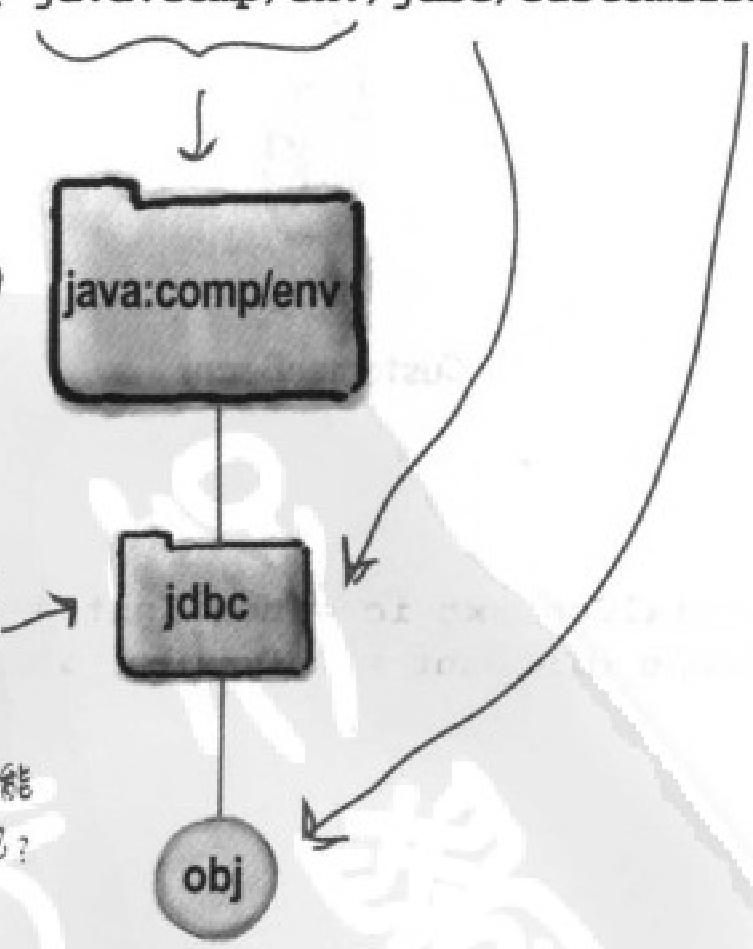
这个bean类型的私有环境

这个子上下文按惯例命名为"jdbc"（不过不一定非得是"jdbc"，说是java:comp/env/CustomerDB也是可以的）

要查找的"东西"的名字（在这里，就是要查"CustomerDB"数据库的资源连接工厂）

bean的业务方法中：

```
InitialContext ic = new InitialContext();
DataSource ds =  (DataSource) ic.lookup("java:comp/env/jdbc/CustomerDB");
Connection conn = ds.getConnection();
```




导航总是从这个上下文开始。所以当bean在它的环境中查东西时，总会在代码中的某个地方看到"java:comp/env"。

规范建议对所有JDBC资源管理器连接工厂都使用"jdbc"子上下文，对邮件连接都使用"mail"子上下文，你应该能猜出url连接用什么子上下文吧？

**"CustomerDB"客户数据库连接工厂的一个引用……是这样吗？**

程序员到底怎么知道数据库的名字？？




## BRAIN POWER

这个问题如何处理呢？程序员如果不知道服务器上数据库是如何配置的，他怎么硬编码一个诸如“java:comp/env/jdbc/CustomerDB”的查找串呢？

[提示：还记得前面讨论安全的时候是怎么做的吗？bean提供者怎么利用isCallerInRole（“someRole”）以程序方式实现安全的？]

# 很简单……

# 如果程序员在代码中放一个自造的JNDI名，他必须在部署描述文件中告诉部署人员。

除非bean提供者做了下面的事情，否则部署人员不会知道bean提供者在代码中放了什么：

A. 直接告诉他。
B. 在即时贴上写上JNDI名，把它贴在部署人员的监视器上。

或者

C. 在部署描述文件中声明引用，并提供有帮助的描述，让部署人员很清楚该为程序员自造的这些名字映射什么。




为部署人员在部署描述文件中声明自造的JNDI名。




把程序员自造的/虚构的名字映射为真正的JNDI名（资源在服务器上就是以这个名字部署或配置的）。

## 环境项：部署时常量

假设你在为一个在线购物车系统编写一个简单的结账bean。你不知道bean最后放在哪里。而且即使你知道，你也很清楚像税率和折扣策略之类的东西很可能会变，就算是同一家公司这也不是固定的。

利用环境项（environment entry），你可以在编写代码的时候使用一个变量，这个变量在运行时能利用一个JNDI查找来填入实际的值。要记住，由bean提供者来选择名字，而由部署人员填入值。

这是查找的一部分，必须放在部署描述文件中。

bean的业务方法中：

```
InitialContext ic = new InitialContext();
Double dbl =  (Double) ic.lookup("java:comp/env/smartCustomerDiscount");
customerDiscount = dbl.doubleValue();
// 使用基本double值计算折扣
```

不要把"java:comp/env/"部分放到部署描述文件中。绝对不要！在部署描述文件中，只放"java:comp/env/"之后的部分。

部署描述文件中

```
<entity>
   ...
   <env-entry>
     <description>discount for smart customers</description>
     <env-entry-name>smartCustomerDiscount</env-entry-name>
     <env-entry-type>java.lang.Double</env-entry-type>
     <env-entry-value>0.05</env-entry-value>
   </env-entry>
   ...
</entity>
```

如果你还有点怜悯之心，应该给部署人员提供一个描述。

程序员选了这个名字

类型必须是String或者是一个包装器类



对于bean提供者来说，这个值是可选的，不过bean提供者可以使用这个元素提供一个默认值。不过在部署这个bean之前，部署人员必须确保它有一个合法的值。

环境项与其他定制内容有所区别，因为部署人员不用从bean提供者给出的名字映射到另外某个实际的名字。在bean提供者说环境项存在之前（也就是把环境项放在<env-entry>中），环境项是不存在的。只要部署bean时<env-entry>有一个值，就会创建这个环境项。

**不能动态地改变一个值！**

一旦bean已经部署，就不能再改变环境变量的值了！要想更新bean看到的值，惟一的办法就是基于新的部署描述文件重新部署这个bean。

## there are no Dumb Questions

**问：** **为什么不能直接使用Java性质文件，而要使用环境项呢？**

**答：** 这里不允许使用性质，因为你不能访问文件系统来读入性质，而且对于系统性质也无从控制（例如，没有办法把一个性质设置为JVM命令行参数）。

**问：** **为什么不能让一个环境项由同一个应用中的多个bean共享呢？或者，至少在同一个ejb-jar中共享？**

**答：** 因为……反正就是不能这样。根本没有共享bean环境的机制，原因是，这样一来，让bean有自己的私有空间就没有多大用处了。bean有自己的环境可以避免一个很有可能出现的灾难：不同bean之间出现命名冲突！换句话说，bean有私有空间，这样不同的bean部署时有同名的环境项（或其他引用）也不会冲突。

如果有一个共享资源或环境项，就必须对部署的每个bean都配置一次。不是每个部署描述文件配置一次，而是每个bean都要单独配置，而不论bean放在哪里。




**必须知道环境项的作用范围！**

你要知道环境项对于每个home是私有而且惟一的。如果你为Customer bean部署了一个值为10的环境项“foo”，那么Customer bean的所有实例从一个业务方法使用java:comp/env/foo查找时都会得到同样的值（10）。

**但是如果你的环境项为“foo”，而且另外某个bean类型也部署了同名的环境项，此时不会导致命名冲突。** Customer bean的“foo”与Advice bean的“foo”是不一样的，就算它们的查找串相同也没关系：ic.lookup(“java:comp/env/foo”)。而且即使两个bean都在同一个ejb-jar或企业归档文件（.ear）中，也不会冲突。

不过这还不算完……一个bean类型部署时不能有多个同名的资源。所以如果在Customer bean中把一个环境项命名为“foo”，就不能同时有一个名为“foo”的资源管理器连接工厂。不过，只要使用子上下文，就没事了。因为“jdbc/foo”与单独的“foo”是不同的。所以可以同时有：

java:comp/env/foo

和

java:comp/env/jdbc/foo

但是若两个资源都名为java:comp/env/foo则不可以，就算它们是不同的资源类型也不行。

**环境项必须是以下的某种类型：**

- String
- Byte
- Short
- Integer
- Long
- Float
- Double
- Character
- Boolean

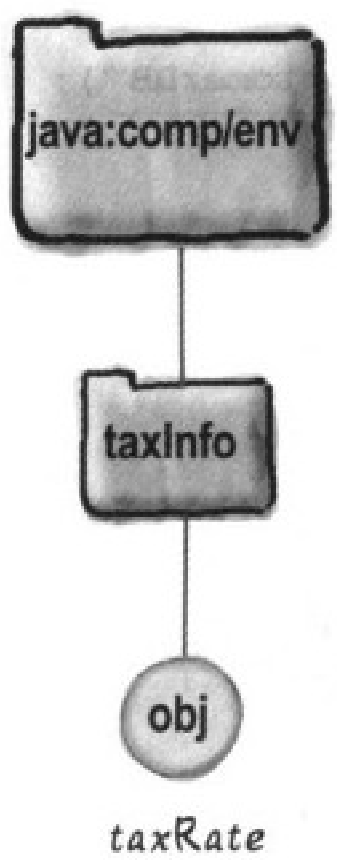
**环境项只能是String和包装器类**

## 关于子上下文

要记住，java:comp/env空间只是一个子上下文（subcontext）。当然这个空间很特殊，但它仍然是一个上下文。所以可以将其保存在一个Context变量中，这样每次想查东西的时候就不用反复地键入“java:comp.env”了。

**保存你的特殊环境上下文：**

```
InitialContext ic = new InitialContext();
Context mySpecialPlace = (Context) ic.lookup("java:comp/env");
// 现在对mySpecialPlace做查找
Double dbl =  (Double) mySpecialPlace.lookup("smartCustomerDiscount");
```




**使用子上下文**

taxInfo是bean环境的一个子上下文

```
InitialContext ic = new InitialContext();
Context myTaxInfoCtx = (Context) ic.lookup("java:comp/env/taxInfo");
// 现在对myTaxInfoCtx子上下文做查找
Double dbl =  (Double) myTaxInfoCtx.lookup("taxRate");
```

注意：除了桩（远程home接口引用），不用对JNDI查找得到的结果窄化，所以只需一个普通的类型强制转换就足够了。

taxRate是taxInfo子上下文中的一个对象。

**创建子上下文**

子上下文之所以存在，只是因为你做了指定。如果在查找代码中键入了“java:comp/env/foo/bar”，就是在说：在我的环境中有一个名为“foo”的子上下文，它包含对象“bar”。 部署这个bean时只要在部署描述文件中使用同一个子上下文，这个子上下文就会神奇地存在，这只是因为你说它是存在的。换句话说，你不用再把整个过程走一遍（创建一个新的JNDI上下文，再对它命名）。想让它在那里，它就在那里。难道你不想这样吗？

在部署描述文件中，环境项名中斜线后面的内容都会自动地变成一个子上下文！

```
<env-entry-name>taxInfo/taxRate</env-entry-name>
```

把“taxInfo/”放在“taxRate”前面，这样在部署这个bean时会自动地创建taxRate子上下文。

## 资源管理器连接工厂
## （请考虑：数据库）

任何bean都可以使用数据库。实际上，即使是使用容器托管持久存储的实体bean也可以得到一个数据库的连接，只要它使用这个数据库不是为了管理它自己的持久存储，而是做其他事情就行。代码很简单：查找一个DataSource，问它要一个Connection。一旦有了一个Connection，就可以发送JDBC语句来执行你的SQL了。

尽管javax.sql.DataSource是目前最常用的资源管理器连接工厂，但是也可以使用其他类型的资源管理器连接工厂，如mail或URL连接。

查找的这一部分必须放到部署描述文件中

**bean的业务方法中：**

```
InitialContext ic = new InitialContext();
DataSource ds =  (DataSource)ic.lookup("java:comp/env/jdbc/CustomerDB");
Connection conn = ds.getConnection();
// 使用连接执行JDBC
```

不要加上java:comp/env/部分，要从第一个子上下文开始(jdbc)

**部署描述文件中**

```
<entity>
   ...
    <resource-ref>

      <res-ref-name>jdbc/CustomberDB</res-ref-name>

      <res-type>javax.sql.DataSource</res-type>

      <res-auth>Container</res-auth>

      <res-sharing-scope>Shareable</res-sharing-scope>

    </resource-ref>

   ...
</entity>
```

程序员自造的名字，而不是实际的数据库名！

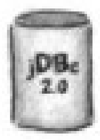
资源管理器连接工厂类型

由谁检查用户是否有权访问数据库？可以是Application或Container

<res-sharing-scope>是可选的，默认值是"Shareable"。这表示同一个应用中使用相同资源并在同一个事务中的其他bean可以使用同一个连接。如果要禁用共享，这个值应当是"Unshareable"。

## 资源管理器连接工厂类型

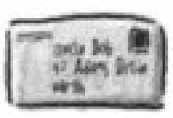
不能在部署描述文件中放任意的资源类型。对于4个标准资源管理器连接工厂（如果你把JMS主题和队列算成是两个不同的类型，那就是5个），必须使用以下的某种类型：



① 
```
<res-type>javax.sql.DataSource</res-type>
```



② 
```
<res-type>javax.jms.QueueConnectionFactory</res-type>
<res-type>javax.jms.TopicConnectionFactory</res-type>
```



③ 
```
<res-type>javax.mail.Session</res-type>
```

http://www.headfirst.com

④ 
```
<res-type>java.net.URL</res-type>
```

尽管只有这4个是EJB 2.0中的标准类型，但是如果你需要访问其他资源（如既有系统），还可以使用连接器（Connector）体系结构。连接器超出了本书的范畴（考试中也不会考这个内容），所以你大可放心。不过，如果需要的话，你应当知道有这样一个连接器体系结构可以使用。

## 资源授权

要向EJB服务器本身完成认证，但这只是一方面，很有可能数据库也有自己的登录机制。用户可能需要另外一个登录名和口令，这与用户向EJB服务器认证时所用的登录名和口令也许是不同的。

作为一个bean提供者，要为资源管理器（如数据库）提供用户的登录数据，有两种方法可供选择。

① 
```
<res-auth>Container</res-auth>
```

容器授权意味着部署人员必须为资源管理器配置登录信息。这完全取决于开发商和资源，而且可能意味着部署人员必须将EJB安全中使用的主体和角色映射到资源管理器所需的角色。至少，部署人员可能要指定一个可以让某个人进入的名字和口令。

② 
```
<res-auth>Bean</res-auth>
```

bean授权意味着程序员要使用getConnection()的重载版本（取一个用户名和一个口令作为参数）：

```
Connection conn = ds.getConnection(userName, password);
```

## 完整的资源映射图

虚构的JNDI名：
<resource-ref>**CustDB**</resource-ref>　实际的JNDI名：**CustDatabase**　实际的数据库名：**CustomerData**

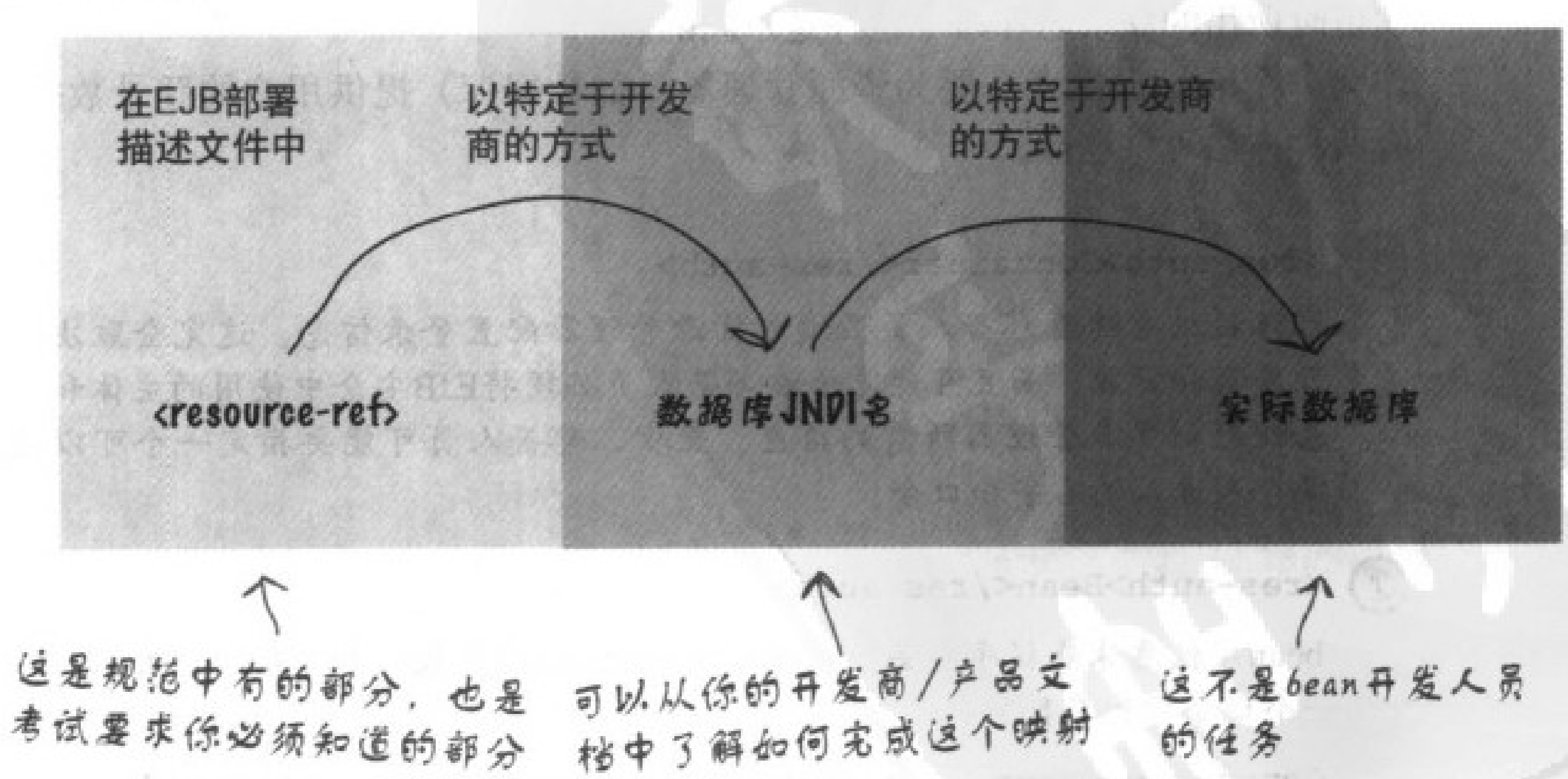



在代码中，bean提供者对DataSource（用于得到一个数据库连接）做了一个查找。他不知道数据库的实际JNDI名（更不知道实际的数据库名），所以他自造了一个。不过他在部署描述文件中声明了一个<resource-ref>元素来告诉部署人员他做了什么。

部署人员把bean提供者自造的<resource-ref>名映射到DataSource注册时所用的实际JNDI名。

系统管理员（或操作环境中的某个人）在服务器上配置数据库，给它一个JNDI名。

### 在哪里映射，如何映射



## EJB引用

（当一个bean需要另一个bean时）

如果bean需要其他的bean，这是服务器中最幸运的bean了。不过要记住，bean像每个人一样，必须通过home接口。如果bean A想在bean B的home上完成一个JNDI查找，我们就会像以往一样遇到同样的问题：bean B的JNDI名是什么？作为一个bean提供者，你可以造一个名字。在部署时，部署人员会把你虚构的名字映射到真正的JNDI名，它与你在部署描述文件中指定的bean类型匹配。

"ejb/AdviceGiver"必须放在部署描述文件中（不能加引号）。记住，编写代码时你并不知道实际的名字会是什么……

**bean的业务方法中：**

```
InitialContext ic = new InitialContext();
Object o =  ic.lookup("java:comp/env/ejb/AdviceGiver");
AdviceHome home = (AdviceHome) PortableRemoteObject.narrow(o, AdviceHome.class);
Advice advisor = home.create();
// 调用Advisor上的方法
```

**部署描述文件中**

千万别加java:comp/env/部分，要从第一个子上下文开始（ejb）

```
<entity>
   ...
   <ejb-ref>

      <ejb-ref-name>ejb/AdviceGiver</ejb-ref-name>

      <ejb-ref-type>Session</ejb-ref-type>

      <home>headfirst.AdviceHome</home>

      <remote>headfirst.Advice</remote>

   </ejb-ref>
   ...
</entity>
```

程序员自造的名字，而不是bean实际的JNDI名。（指向 <ejb-ref-name>）

这必须是Session或Entity（指向 <ejb-ref-type>）

home接口的完全限定名（指向 <home>）

组件接口的完全限定名（指向 <remote>）

## EJB本地引用

如果你想查找bean的本地home，要使用<ejb-local-ref>标记。

对于本地接口，子元素有所不同：这里是<local-home>而不是<home>，另外这里是<local>而不是<remote>

```
<ejb-local-ref>
    <ejb-ref-name>ejb/AdviceGiver</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>headfirst.AdviceHomeLocal</local-home>
    <local>headfirst.AdviceLocal</local>
</ejb-local-ref>
```

### there are no Dumb Questions

**问：** <home>和<local-home>这两个元素名我能理解，这是有道理的。但是<remote>和<local>是怎么回事?应该是 <component>和<local-component>才对吧?

**答：** 你可以放眼现在，但是过去也不容忘记。在EJB 2.0之前，并没有本地接口的概念。所以客户视图总是称为Home和Remote。尽管当时这个命名机制已经存在不一致了（因为home和业务方法接口都是Remote（java.rmi.Remote）），但是等到本地接口登场时，情况就变得更加复杂。我们来看看……可以把本地home接口命名为“local-home”，再把本地remote接口命名为“local-remote”。发现问题了吗?所以在EJB 2.0中，我们不再把业务接口称为“远程接口”，而是改叫成“组件接口”。现在我们把这两个接口称作“本地组件接口”和“远程组件接口”。

**问：** 嗯，你并没有回答我的问题。为什么远程组件接口的标记还是“remote”，本地组件接口的标记还是“local”？

**答：** 这还是因此过去遗留的问题（为了向后兼容等等）。在EJB 1.1中，<ejb-ref>标记用的是<home>和<remote>，所以现在还是这样。这个命名机制基本上是这样：组件接口可以是本地或远程的。如果标记没有明确地说“home”，那说的就是组件接口。这些小小的不一致性正是EJB魔力的一部分。

**<res-ref-type> 不能是消息驱动bean!**

<ejb-ref-type>元素可以是“Session”或“Entity”。只能如此。一个人（比如说一个bean）怎么能查找一个消息驱动bean呢？要记住，消息驱动beans根本没有客户！在JNDI中根本没有要查的接口，所以不要犯糊涂……

**不要在<ejb-ref>中指定bean类型。**

但是往往会认为应该这么做。你要放入bean的home接口和组件接口类型，而不是bean类。为什么不指定bean类型呢？想想看不指定bean类型所带来的灵活性。再想想如果bean提供者必须知道bean的确切的类类型，bean提供者和部署人员的关系是不是太过密切了！

## 对EJB引用使用<ejb-link>

如果应用组装人员看到一个bean有同一个应用中另一个bean的EJB引用，就应该使用 <ejb-link> 把<ejb-ref> 链接到部署描述文件中指定的另一个bean。可以把<ejb-link>想成是一种“跳到这个标签”的跳转标记，这里链接的值与别处一个<ejb-name>元素的值匹配。

**<ejb-link>值必须与另外某个bean的<ejb-name>值匹配。**

**要记住：<ejb-name>只是部署描述文件中的一个标签。它不必与类名、接口名、JNDI名或其他名字匹配。它只是部署描述文件中对应一个特定bean的标签。**

**就算你把bean按你的宠物来取名，除了你的同事，没有人会在乎。**

**在部署描述文件中的某个地方**

```
<entity>
    ...
    <ejb-ref>
        <ejb-ref-name>ejb/AdviceGiver</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>headfirst.AdviceHome</home>
        <remote>headfirst.Advice</remote>
        <ejb-link>AdviceEJB</ejb-link>
    </ejb-ref>
    ...
</entity>
```

<ejb-link>必须匹配一个<ejb-name>值（这个部署描述文件中（或者同一个J2EE应用中的另一个部署描述文件）另外某个bean的<ejb-name>值）。

**在部署描述文件中的另一个地方**

```
<session>
    <ejb-name>AdviceEJB</ejb-name>
    ...
</session>
```

记住，<ejb-name>只是部署描述文件中的一个标签，以便让部署描述文件中的其他部分引用。这不是实际的bean类名之类的东西（除非你碰巧把ejb-name就取名为bean类名）。

**<ejb-name>在一个部署描述文件中（这意味着一个ejb-jar）必须惟一，而不是在一个J2EE应用中惟一。**

这并不意味着同一个J2EE应用中的所有bean都必须有惟一的 <ejb-name>值。一个 .ear 可以有多个ejb-jar文件，每个ejb-jar有一个部署描述文件，而且应用中的各个部署描述文件可以有相同的<ejb-name>。真让人高兴……不过，这样一来<ejb-link>就有问题了，它要在整个应用中（而不只是当前部署描述文件中）寻找匹配的<ejb-name>。不用担心。如果你的应用中确实有两个包括相同<ejb-name>项的部署描述文件，可以使用另一个<ejb-link>语法，要向ejb-jar增加路径，也就是与这个链接匹配的<ejb-name>所在的路径。路径（“#”之后的部分）相对于当前部署描述文件的jar。

```
<ejb-link>../custServices/advice.jar#AdviceEJB</ejb-link>
```

## 资源环境引用（可以考虑：JMS目标）

作为一个bean提供者，可以在JNDI中查找两种与资源有关的东西：一种是资源管理器连接工厂引用，另一种是所谓的受管对象（administered object）的引用。其主要区别在于，资源环境引用是你想要的一个东西的引用，而工厂给你的只是你想要的东西的一个连接。换句话说，受管对象是一个目标，而资源管理器连接工厂引用只是得到你想要的东西的第一步（一个连接）。

不过现在你要换换脑子，只要看到资源环境引用，就把它换成JMS目标，因为如今使用的资源环境引用主要就是JMS目标。不错，以往正是把资源环境引用叫作“JMS目标引用”，但是从长远考虑这样过于限制了。再说了，这样太明确了，显得没有深度。

bean的业务方法中：

```
InitialContext ic = new InitialContext();
Object o =  ic.lookup("java:comp/env/jms/NewCustomerQueue");
javax.jms.Queue custQ = (javax.jms.Queue) o;
// 使用custQ
```

不要加java:comp/env/部分，而是要从第一个子上下文开始（jms）（你可能已经听烦了）

部署描述文件中

```
<entity>
   ...
   <resource-env-ref>
      <resource-env-ref-name>jms/NewCustomerQueue</resource-env-ref-name>

      <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>

   </resource-env-ref>
   ...
</entity>
```

程序员自造的名字，而不是Queue真正的JNDI名 ⟶ （指向 resource-env-ref-name）

这也可以是javax.jms.Topic（不必限制为只使用JMS目标，不过目前来讲只有这一个标准）

# 关于部署描述文件，bean提供者和应用组装人员各自的职责

不要担心，现在不用把所有这些都记住！在后面逐步介绍这些内容的时候，你就会更清楚。至于现在，只要对他们各自负责些什么有一个总体的概念就可以了。



与bean类中代码相关的内容主要由Bill放入



关于应用中两个或多个bean有何关系的内容主要由Annie放入，有时她还要为一个特定应用定制bean信息。

## bean提供者

- bean名
- bean类、home接口和组件接口的完全限定名
- bean类型（会话bean、实体bean等等）
- 再入（仅用于实体bean）
- 会话bean的状态管理（无状态还是有状态）
- 事务划分类型（bean托管还是容器托管）
- 实体bean持久存储管理（bean托管还是容器托管）
- 主键类
- 对于CMP：抽象模式名、CMP字段、CMR关系、查找方法和选择查询。
- 资源管理器连接工厂引用
- 环境项声明
- EJB引用（本地和远程）
- 安全角色引用
- 对于消息驱动bean：目标、消息选择方法以及确认模式。

## 应用组装人员

修改bean提供者的信息：

- 环境项的值
- 描述字段（修改或新建）
- 关系名修改
- 消息驱动bean消息选择方法（可能是有所限制，但不是替换）

应用组装信息（都是可选的）：

- 绑定企业bean引用（也就是说，把一个bean与同一个ejb-jar或J2EE应用中的另一个bean链接起来）
- 安全角色（bean客户的推荐角色）
- 方法许可：安全角色与beanhome接口和组件接口中方法之间的一个关系。
- 将安全角色引用链接到安全角色
- 安全身份类型：调用者（caller）还是run-as
- CMT bean方法的事务属性

# 关于部署描述文件，部署人员的职责



部署人员

修改bean提供者的信息：

- 确保所有环境项都有合法值。

其他与部署描述文件有关的任务。所有这些都是用一个特定于开发商的工具完成，而不是 ejb-jar部署描述文件的一部分：

安全

- 指定应用的安全领域（security domain）和主体域（principal realm）。
- 为安全角色（而不是安全角色引用）指定主体和/或组。
- 为组件间的调用委托主体（也就是说，配置run-as主体）。

资源管理器连接工厂

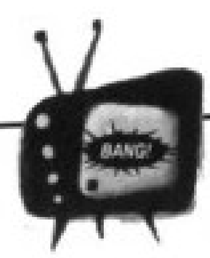
- 将资源管理器连接工厂引用绑定到操作环境中一个实际的资源管理器连接工厂。
- 为容器授权的资源访问配置登录信息。

EJB引用

- 确保所有EJB引用都与操作环境中已存在bean的home绑定。
- 确保目标bean与EJB引用中声明的类型是类型兼容的。

资源环境引用（JMS主题或队列）

- 确保所有声明的资源引用都绑定到操作环境中存在的对象上，并确保目标对象与所声明的类型是类型兼容的。




**部署人员使用特定于开发商的工具来做工作**

关于部署人员和他对部署描述文件所做的工作，你可能会发现一个问题，心里想："部署人员根本没有动部署描述文件中的任何东西，所以这没部署人员的任务。"不过……关于部署描述文件中的许多东西，部署人员确实有许多职责，所以请仔细看任务的描述。例如，谁把安全角色映射到安全角色引用？是应用组装人员。那么谁把主体映射到安全角色呢？是部署人员。

# 记住谁做什么

想想看…… 我怎么可能知道运行这个bean的服务器上那些东西都是些什么名字呢？我编写代码的时候，并不知道bean最后会在哪里！所以如果我的bean使用了另一个bean，我只能猜这个bean的安全角色名或JNDI名是什么。



bean提供者

只有我才知道服务器是怎么配置的，以及所有一切的JNDI名。也只有我知道这个公司的安全领域。但是我不清楚代码里有什么，所以我要靠bean提供者告诉我（通过部署描述文件）他“自造”的名字，这样我才能把这些虚构的名字映射到服务器上的真正名字。

部署人员



我必须集成两个或多个bean（可能来自不同的开发商）。所以，如果bean A使用了自造的安全引用“Employee”，而bean B使用了自造的安全引用“Minions”，我必须把它们都映射到一个安全角色“Slaves”。bean提供者在部署描述文件中声明了自造的名字。这样就告诉了我，他在代码中做了什么。

应用组装人员



# 下面来看bean的运行时环境

我们几乎介绍完了bean的世界。既然已经了解了bean的特殊JNDI环境，关于bean的运行时环境还有一些细节需要了解。这些内容考试都有要求，所以现在可别打瞌睡。就快学完了，曙光就在前面！

- **保证有的API**

  必须清楚哪些API是每一个EJB 2.0容器都保证要有的，而哪些不是EJB 2.0容器的一部分。例如，JMS就是支持的，而JXTA不是。

- **保证有的服务**

  必须知道每一个EJB 2.0容器都保证提供哪些支持，而哪些不能保证。例如，事务支持可以保证，而负载平衡不一定支持。

- **ejb-jar的结构**

  可能这并不算是运行时环境，但是再没有哪个地方适合说ejb-jar的结构了，而你又必须知道这个内容。所以我们把它放在这里，万一你忘记了我们在第一章讲的东西，可以在这里重温一下。例如，你必须知道一个ejb-jar可以没有一个清单，但是必须有一个META-INF目录，而且这个目录下必须有部署描述文件。这个部署描述文件的文件名必须是“ejb-jar.xml”。

- **编程限制**

  如果你希望你的bean在任何EJB 2.0兼容容器上都是可移植/兼容的，就不能做下面所列的事情，就算你的开发商允许那样做也不行（开发商可能不是有意的）。而这还不只是为了可移植性，这些限制还与安全有关。例如，如果你想管理自己的线程，就会挡容器的道，容器知道你最后会搞得一团糟。你肯定不希望对容器的工作瞎搅和吧。

  如果你的容器允许这么做，仅凭这一点并不能说明这样做是对的。考试时，你必须知道规范中对哪些做了限制。如果考试时问你某种情况是否可能，而它是规范中明确限制的，就算你的开发商允许这样做，你也必须说不可能。对考试来说，如果你做了一件事后不能保持可移植性，这件事情就是不能做的。

# 哪些API是EJB 2.0保证支持的？

## 所支持的API

- Java 2标准版, v 1.3 (J2SE)
- JNDI API 1.2
- JTA 1.01扩展（仅UserTransaction接口）
- JDBC 2.0扩展
- JMS 1.02扩展
- JavaMail 1.1，仅发送邮件
- JAXP 1.0

**J2SE支持的是Java 1.3，而不是1.4!**

如果一个API在1.4版本的J2SE标准库中，这并不意味着EJB保证支持这个API！EJB规范只要求J2SE的1.3版本，所以如果你编写的一个bean要依赖于J2SE 1.4中的某个东西，这个bean就不能保证在所有EJB 2.0容器上可移植。

例如，这说明尽管Java加密扩展包（Java Cryptography Extension, JCE）现在是J2SE平台1.4版本的一部分，但不能保证EJB 2.0一定支持它。

**Relax** **不用记住准确的版本号**

考试中不会出有关版本号的那些小题目，比如JavaMail是1.1还是1.2，或者JTA扩展是1.01还是1.03。除了EJB 2.0中的2.0，你只要知道JDBC 2.0扩展包，还有只能保证支持J2SE 1.3版本（不一定支持1.4）就行了。

**Sharpen your pencil**

不看上面，你知道这些API各自是干什么的吗？如果你不知道，可以根据你对EJB的了解猜猜看。

我们已经把最难的一个给你做出来了。

| | |
|---|---|
| J2SE | ______ |
| JNDI | ______ |
| JTA | ______ |
| JDBC | ______ |
| JMS | ______ |
| JavaMail | 发送邮件 |
| JAXP | ______ |




## Make it Stick

Bob是JMS送信人（messenger），他要送一张图，这个图上有一个JDBC罐子，罐子的上面是一封Java-Mail，罐子的下面是一个JNDI命名盒子。

他把这张图送给一群Jax (P)。

好在，他送信受一个Java事务API（Java Transaction API，JTA）管理，一切正常，所以他可以尽情地享受一个1.3小时的午餐。

（牢牢记住这个故事和这个图，这样你在考试的时候能更容易地想起来。）




### EJB规范里没有的API

要当心不能保证的API。有些API（如Servlet或JSP）在J2EE规范中有保证，但是EJB规范不能保证，我们的考试只关心EJB规范里有的东西。

哪些在EJB 2.0中没有呢?下面这些可能会考：

JXTA
JTS
Jini
JMX
JCA
JSP
Servlet

EJB 2.0不能保证支持这些API!!

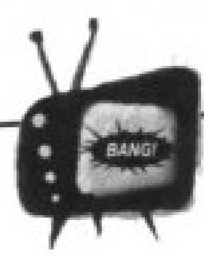
这次送信由Java事务API(JTA)管理!



## 一些重要的、有保证的服务和行为：

- 分布式**事务**
- **线程安全**
- 实体bean的容器托管**持久存储**
- **安全领域**和一个主体域（不能保证多个主体域）
- 保证部署描述文件和其他部署工具指定的**客户访问**安全策略
- 实现提供给bean的java:comp/env环境命名上下文
- **生成**实现home和组件接口的类，以及生成远程对象的桩类。
- 为服务器上配置的资源**实现**资源管理器连接工厂类。




**有些东西听上去很不错，但没有保证！**

每个人都在谈论他们的集群、故障恢复和负载平衡系统。不错，这些系统可以完成故障恢复、实体数据的懒加载，还有内存中数据缓存。

尽管大多数J2EE开发商都提供了一个或多个此类功能，但规范中未做保证！

一定要清楚保证与不能保证之间有什么区别。请看规范中“容器提供者职责”部分。

# ejb-jar中要必须有些什么？




# ejb-jar的结构




**必须知道哪些不在ejb-jar中。**

你要知道ejb-jar文件中没有什么。注意到这里少了什么？哪个东西是你认为JAR里应该有但实际上没有的？给你一个提示：它是通常放在META-INF目录里的东西。

知道了吗？正是清单！EJB 2.0不要求有清单文件。你可以放一个清单进去，但这是可选的，而且通常都不需要。

你还必须知道ejb-jar不能有哪些东西：容器生成的类不能放在里面！记住，容器实现了home接口和组件接口，如果接口是远程的，还会建立桩。在部署之前这是不会发生的……所以它们不是ejb-jar文件的一部分，不能由bean提供者或应用组装人员放在他们的交付文件里。

**不要求遵循命名约定！**

bean的命名约定是把组件名作为组件接口名，并增加“home”或“bean”分别作为home接口和bean类名。我们强烈建议你遵循这个约定：这样可以使部署容易得多，别人也能更轻松地读你的代码。但是这不是一个严格的要求。

## 编程限制

如果想保证你的bean在任何EJB 2.0兼容的服务器上都可以部署，那么EJB中哪些是要避免的？

- ServerSocket
- 获得安全策略信息
- 读/写静态字段
- 加载一个本地库
- AWT输出或键盘输入
- 线程
- 创建或获得一个类加载器
- java.io包

**不要上当，有些东西看上去不好，但实际上没有什么**

你要知道在EJB 2.0中能做什么，不能做什么。如果你想得太多，可能就会多虑，认为（考试中）什么都不对。

下面这些在EJB中是可以做的，考试时可能会考到：

**java.net.Socket**——普通的客户Socket是可以的。只是不能监听ServerSocket。

**让bean类扩展另一个类**——对于bean类来说，正常的java继承是可以的。

**把一个bean字段同时标记为静态（static）和最终（final）**——实际上，这是标记一个静态字段的惟一办法。（见不允许读/写静态字段的限制。）

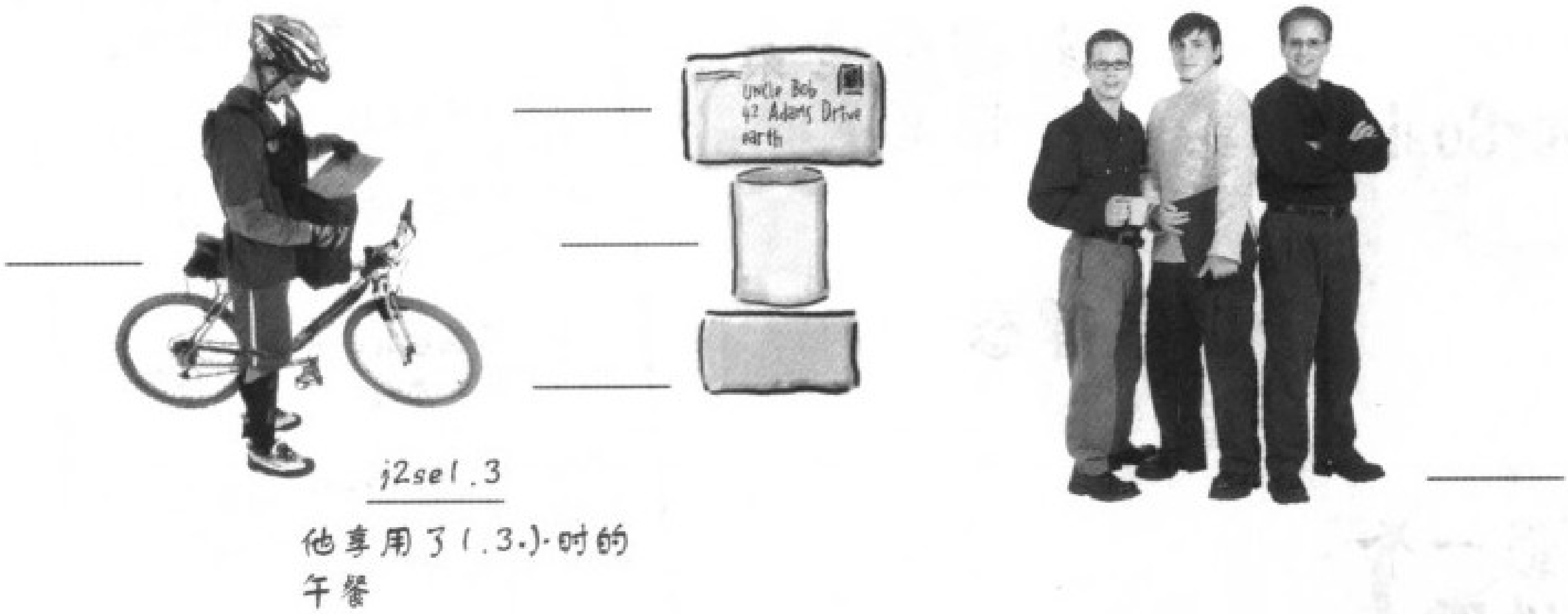
### BRAIN POWER

你肯定想记住这些编程限制。最好的办法是什么？先停下来，好好想想这些不允许做的事情，再说说为什么存在这个限制，要说出一个或几个理由来。想清楚以后，再去看EJB 2.0规范中的第494页，在那里对这些限制（和其他一些限制）做了描述。




# 一定要记住

① 请看下面的图，看你能不能讲出这个故事，在合适的地方放上相应的API。我们已经帮你填了一个空。




---

② 用以下给出的各个部分（而且只能用这些部分），把它们重新组装为正确的配置（根据需要画线）。在右边的空白处画出最终的结构，并在目录上写上正确的名字，给出.xml文件的文件名。这个bean在com.headfirst包中。

在这里画出JAR文件的结构：



# 模拟测验

**1** 哪些API是EJB 2.0容器保证支持的？（选出所有正确的答案。）

❑ A. JAXP

❑ B. JNDI

❑ C. JXTA

❑ D. JDBC

❑ E. JMS

**2** 关于企业bean的环境，以下哪种说法是正确的？（选出所有正确的答案。）

❑ A. 环境项对于同一个企业bean类型的实例是惟一的

❑ B. 在一个EJB 2.0容器中，一个EJB可以有多组环境项

❑ C. EJB的环境项的值可以由EJB在运行时修改

❑ D. 环境项值可以是基本类型或包装器类型

**3** 哪些API是EJB 2.0容器保证支持的？（选出所有正确的答案。）

❑ A. J2SE 1.3

❑ B. JAXB 1.0

❑ C. JAXR 1.0

❑ D. JAXP 1.0

**4** 给定一个名为“Customer”的bean，以及一个名为“lastName”的环境项，以下哪个bean类代码段会返回这个环境项的值？（选出所有正确的答案。）

❑ A.
```
Context c = new SessionContext();
Context e = (Context) c.lookup( "java:comp/env" );
String name = (String) e.lookup( "lastName" );
```

❑ B.
```
Context c = new InitialContext();
Context e = (Context) c.lookup( "java:comp/env/
Customer" );
String name = (String) e.lookup( "lastName" );
```

❑ C.
```
Context e = new Lookup( "java:comp/env" );
Context c = new InitialContext(e);
String name = (String) c.lookup( "lastName" );
```

❑ D.
```
Context c = new InitialContext( "Customer" );
Context e = (Context) c.lookup( "java:comp/env" );
String name = (String) e.lookup( "lastName" );
```

❑ E.
```
Context c = new InitialContext();
Context e = (Context) c.lookup( "java:comp/env" );
String name = (String) e.lookup( "lastName" );
```

**5** 编写一个会话bean类时，要保证这个bean在所有EJB 2.0容器上都可移植，应避免哪些技术？（选出所有正确的答案）。

❑ A. 使用`java.net.Socket`类

❑ B. 使用内部类

❑ C. 对字段使用“final”修饰符

❑ D. 将“this”作为参数传递

**6** 在一个bean的部署描述文件中，以下哪个类型是**<env-entry-type>**元素中的合法数据类型？（选出所有正确的答案。）

- ❑ A. **byte**
- ❑ B. **short**
- ❑ C. **ArrayList**
- ❑ D. **java.lang.Boolean**
- ❑ E. **java.lang.Character**

**7** 编写EJB时，要保证这个EJB在所有EJB 2.0容器上都可移植，应避免哪些声明？(选出所有正确的答案。)

- ❑ A. **final int x;**
- ❑ B. **static int x;**
- ❑ C. **final static int x;**
- ❑ D. **final transient int x;**

**8** 通常由谁负责在bean的部署描述文件中指定查找文件和选择查询？

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 系统管理员
- ❑ E. 服务器提供者

**9** 要得到一个JDBC连接，会使用哪个（哪些）部署描述文件元素？（选出所有正确的答案。）

- ❑ A. **<ejb-ref>**
- ❑ B. **<ejb-link>**
- ❑ C. **<role-name>**
- ❑ D. **<env-entry>**
- ❑ E. **<resource-ref>**

**10** 通常由谁将多个ejb-jar文件合并为一个ejb-jar文件？

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 系统管理员
- ❑ E. 服务器提供者

**11** bean提供者要找到其他EJB的home接口，会使用哪个（哪些）部署描述文件元素？（选出所有正确的答案。）

- ❑ A. **<ejb-ref>**
- ❑ B. **<res-type>**
- ❑ C. **<env-entry>**
- ❑ D. **<role-name>**
- ❑ E. **<resource-ref>**

**12** 关于资源管理器连接工厂引用，bean提供者有哪些职责？（选出所有正确的答案。）

- ❑ A. 在EJB服务器上配置资源管理器
- ❑ B. 为资源管理器配置登录信息
- ❑ C. 为部署描述文件指定这样一个引用
- ❑ D. 创建与JNDI的一个符号链接

**13** ejb-jar文件被认为属于哪两方面之间的合约？（选出所有正确的答案。）

- ❑ A. bean提供者和系统管理员之间
- ❑ B. bean提供者和应用组装人员之间
- ❑ C. 应用组装人员和部署人员之间
- ❑ D. 应用组装人员和系统管理员之间
- ❑ E. 部署人员和系统管理员之间

**14** 在每个ejb-jar文件中必须包括哪些类文件（直接或者间接）？（选出所有正确的答案。）

- ❑ A. 企业bean类
- ❑ B. EJBObject接口的桩类
- ❑ C. 企业bean的超类
- ❑ D. 用做参数或返回类型的所有J2SE类

**15** 通常由谁负责声明部署描述文件中的资源连接工厂引用？

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 容器提供者
- ❑ E. 系统管理员

**16** 关于一个合法的ejb-jar文件，以下哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 必须包含一个home接口和一个组件接口
- ❑ B. 部署描述文件是可选的
- ❑ C. 必须包含这个bean所用的所有J2EE类
- ❑ D. 企业bean类是可选的

**17** 通常由谁负责建立资源管理器登录信息？

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 容器提供者
- ❑ E. 系统管理员

# 模拟测验答案

**1** 哪些API是EJB 2.0容器保证支持的？（选出所有正确的答案。） （规范：493）

✔ A. JAXP

✔ B. JNDI

❑ C. JXTA

✔ D. JDBC

✔ E. JMS

**2** 关于企业bean的环境，以下哪种说法是正确的？（选出所有正确的答案。） （规范：410～412）

❑ A. 环境项对于同一个企业bean类型的实例是惟一的

✔ B. 在一个EJB 2.0容器，一个EJB可以有多组环境项

❑ C. EJB的环境项的值可以由EJB在运行时修改

❑ D. 环境项值可以是基本类型或包装器类型 — 只允许是String和包装器类型

**3** 哪些API是EJB 2.0容器保证支持的？（选出所有正确的答案。） （规范：496～497）

✔ A. J2SE 1.3

❑ B. JAXB 1.0

❑ C. JAXR 1.0

✔ D. JAXP 1.0

**4** 给定一个名为“Customer”的bean，以及一个名为“lastName”的环境项，以下哪个bean类代码段会返回这个环境项的值？（选出所有正确的答案。） （规范：411～412）

❑ A.
```
Context c = new SessionContext();
Context e = (Context) c.lookup( "java:comp/env" );
String name = (String) e.lookup( "lastName" );
```
— SessionContext不同于JNDI上下文

❑ B.
```
Context c = new InitialContext();
Context e = (Context) c.lookup( "java:comp/env/
Customer" );
String name = (String) e.lookup( "lastName" );
```

❑ C.
```
Context e = new Lookup( "java:comp/env" );
Context c = new InitialContext(e);
String name = (String) c.lookup( "lastName" );
```

❑ D.
```
Context c = new InitialContext( "Customer" );
Context e = (Context) c.lookup( "java:comp/env" );
String name = (String) e.lookup( "lastName" );
```
— 这里没有参数

☑ E.
```
Context c = new InitialContext();
Context e = (Context) c.lookup( "java:comp/env" );
String name = (String) e.lookup( "lastName" );
```

**5** 编写一个会话bean类时，要保证这个bean在所有EJB 2.0容器上都可移植，应避免哪些技术？（选出所有正确的答案。） （规范：494～495）

❑ A. 使用**java.net.Socket**类 — 客户Socket是可以的，只是ServerSocket不行

❑ B. 使用内部类

❑ C. 对字段使用“final”修饰符

☑ D. 将“this”作为参数传递

**6** 在一个bean的部署描述文件中，以下哪个类型是 **<env-entry-type>**元素中的合法数据类型？（选出所有正确的答案。） (规范:413)

- [ ] A. **byte**
- [ ] B. **short**
- [ ] C. **ArrayList**
- [x] D. **java.lang.Boolean** — 只支持String和包装器类型
- [x] E. **java.lang.Character**

**7** 编写EJB时，要保证这个EJB在所有EJB 2.0容器上都可移植，应避免哪些声明？（选出所有正确的答案。） (规范:494)

- [ ] A. **final int x;**
- [x] B. **static int x;** — static的同时还应当是final
- [ ] C. **final static int x;**
- [ ] D. **final transient int x;**

**8** 通常由谁负责在bean的部署描述文件中指定查找文件和选择查询？ (规范:456~457)

- [x] A. bean提供者
- [ ] B. 应用组装人员
- [ ] C. 部署人员
- [ ] D. 系统管理员
- [ ] E. 服务器提供者

**9** 要得到一个JDBC连接，会使用哪个（哪些）部署描述文件元素？（选出所有正确的答案。） (规范:424)

- [ ] A. **<ejb-ref>**
- [ ] B. **<ejb-link>**
- [ ] C. **<role-name>**
- [ ] D. **<env-entry>**
- [x] E. **<resource-ref>**

**10** 通常由谁将多个ejb-jar文件合并为一个ejb-jar文件？ (规范:458)

- ❑ A. bean提供者
- ✔ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 系统管理员
- ❑ E. 服务器提供者

**11** bean提供者要找到其他EJB的home接口，会使用哪个（哪些）部署描述文件元素？（选出所有正确的答案。） (规范:416)

- ✔ A. **<ejb-ref>**
- ❑ B. **<res-type>**
- ❑ C. **<env-entry>**
- ❑ D. **<role-name>**
- ❑ E. **<resource-ref>**

**12** 关于资源管理器连接工厂引用，bean提供者有哪些职责？（选出所有正确的答案。） (规范:421)

- ❑ A. 在EJB服务器上配置资源管理器
- ❑ B. 为资源管理器配置登录信息
- ✔ C. 为部署描述文件指定这样一个引用
- ❑ D. 创建与JNDI的一个符号链接

**13** ejb-jar文件被认为属于哪两方面之间的合约？（选出所有正确的答案。） (规范:487)

- ❑ A. bean提供者和系统管理员之间
- ✔ B. bean提供者和应用组装人员之间
- ✔ C. 应用组装人员和部署人员之间
- ❑ D. 应用组装人员和系统管理员之间
- ❑ E. 部署人员和系统管理员之间

**14** 在每个ejb-jar文件中必须包括哪些类文件（直接或者间接）？（选出所有正确的答案。） （规范：488）

- ☑ A. 企业bean类
- ❑ B. EJBObject接口的桩类
- ☑ C. 企业bean的超类
- ❑ D. 用作参数或返回类型的所有J2SE类 — 已经有了！

**15** 通常由谁负责声明部署描述文件中的资源连接工厂引用？ （规范：423）

- ☑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 容器提供者
- ❑ E. 系统管理员

**16** 关于一个合法的ejb-jar文件，以下哪些说法是正确的？（选出所有正确的答案。） （规范：488）

- ☑ A. 必须包含一个home接口和一个组件接口
- ❑ B. 部署描述文件是可选的
- ❑ C. 必须包含这个bean所用的所有J2EE类
- ❑ D. 企业bean类是可选的

**17** 通常由谁负责建立资源管理器登录信息？ （规范：422）

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ☑ C. 部署人员
- ❑ D. 容器提供者
- ❑ E. 系统管理员

# 附录A：

# 最后的模拟测验

COFFEE CRAM

在你确信已经充分准备好之前先不要做这个模拟题。如果做得太早，等你下次再做的时候这就不是一份新考卷了，你会对其中的一些题目留有印象，这样会让你得到不符合实际的高分。我们真地很希望你能一次就通过考试。（不过，要是你没能通过考试，每次重考时都再买一次这本书那我们倒是很欢迎……）

为了避免“我记得见过这个问题”的情况，这个模拟测验要比实际考试稍难一点，这里没有告诉你每个问题到底有多少个正确答案。我们的问题和答案从语调、风格、难度到内容都与实际考试几乎一样，但是由于没有告诉你该选择多少个答案，所以你无法根据这个提示排除任何答案。我们确实很严格，不过希望你知道，碰上这种题目对我们自己来说也很棘手，甚至比你更头疼（不过你该庆幸，就在几年前Sun实际的Java考试采用的就是这种题型，大多数问题的后面都要求“选出所有正确的答案”）。

参加考试的大多数人都说我们的模拟测验比实际的SCBCD要难一些，但是他们的测验成绩与实际考试的得分却很接近。这个模拟测验可以很好地评测出你准备得怎么样了，但是要求你做到以下几点：

(1) 必须在2个小时内完成，就像参加实际的考试一样。

(2) 在做模拟测验的时候不能看书！

(3) 不要一遍遍地反复做。等做到第4遍的时候，你可能98%都能做对但仍然未能通过实际考试，这只是因为你记住了这个模拟测验的题目和答案。

(4) 在做模拟测验之前不要喝太多的酒或者其他提神的东西（比如说Ben and Jerry's™ Fudge Brownie、Red Bull™、 Frosted Flakes™ cereal等等）。

# 最后的模拟测验

**1** EJB 2.0有什么好处？（选出所有正确的答案。）

- ❑ A．服务器瘫痪后，MDB还可幸存
- ❑ B．一个实体的表示可以由多个客户共享
- ❑ C．支持嵌套事务
- ❑ D．动态服务恢复
- ❑ E．可以用声明方式设置隔离级别

**2** 哪些方法是客户直接调用的？（选出所有正确的答案。）

- ❑ A. `ejbPassivate()`
- ❑ B. 业务方法
- ❑ C. `setSessionContext()`
- ❑ D. `newInstance()`
- ❑ E. `create()`

**3** EJBHome接口里有哪些方法？（选出所有正确的答案。）

- ❑ A. remove(Handle handle)
- ❑ B. remove(Object primaryKey)
- ❑ C. getHandle()
- ❑ D. 以上都不是

**4** 哪些方法可以由处于入池状态的CMP bean运行？（选出所有正确的答案。）

- ❑ A. **ejbLoad()**
- ❑ B. **ejbFind()**
- ❑ C. **ejbStore()**
- ❑ D. **ejbCreate()**
- ❑ E. 业务方法
- ❑ F. **ejbHome()**

**5** 关于消息驱动bean，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 对消息驱动bean实例的所有调用都必须串行化
- ❑ B. 容器能保证消息会按其接收的顺序得到处理
- ❑ C. bean的 **ejbCreate()** 方法必须取一个类型为 **javax.jms.Message**的参数
- ❑ D. bean提供者使用部署描述文件来指示bean类的实例面向主题还是队列

**6** 关于一个企业bean的环境，哪些说法是正确的？ （选出所有正确的答案。）

- ❑ A. 在bean能访问其环境项之前，这个bean必须先使用一个**SessionContext**对象得到命名上下文
- ❑ B. 只有bean提供者才能设置环境项值
- ❑ C. bean的环境项只能保存在“**java:comp/env**” 或其某个子上下文中
- ❑ D. 对于bean代码中的每个环境项查找，在bean的部署描述文件中都必须有一个匹配的**<env-entry>** 元素

**7** 关于EJB中的安全角色，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 安全角色是使用<security-role>元素在部署描述文件中定义的
- ❑ B. 安全角色的作用范围是实例级的
- ❑ C. 多个方法可以映射到一个安全角色
- ❑ D. 一个方法只能出现在一个安全角色中

**8** 一个EJB 2.0容器必须至少支持哪些API的一个子集？（选出所有正确的答案。）

- ❑ A. JTA 1.0.1
- ❑ B. JDBC 2.0
- ❑ C. JMS 1.0.2
- ❑ D. JAX-RCP 1.0

**9** 关于无状态会话bean，哪些说法是正确的？（选出所有正确的答案。）？

- ❑ A. 可以使用bean托管事务划分
- ❑ B. 必须有一个无参数创建方法
- ❑ C. 一个实例可以支持并发调用
- ❑ D. 必须扩展javax.ejb.SessionBean

**10** 哪个功能只在以下的某一个接口中有（而不是两个接口中都有）

－实体对象的远程组件接口

－实体对象的本地组件接口

（选出所有正确的答案。）

- ❑ A. 删除对象
- ❑ B. 得到对象的句柄
- ❑ C. 调用业务方法
- ❑ D. 得到对象的主键

**11** 创建一个CMP实体bean时，哪些方法是可选的？（选出所有正确的答案。）

- ❑ A. **ejbLoad()**
- ❑ B. **ejbCreate()**
- ❑ C. **ejbRemove()**
- ❑ D. **ejbSelect()**
- ❑ E. **ejbPassivate()**
- ❑ F. **setEntityContext()**

**12** 哪些方法可以在处于入池状态的bean上调用？（选出所有正确的答案。）

- ❑ A. **ejbFind()**
- ❑ B. **ejbLoad()**
- ❑ C. **ejbStore()**
- ❑ D. **ejbSelect()**
- ❑ E. **ejbPassivate()**

**13** 关于消息驱动bean的生命周期，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. **onMessage**方法完成时，容器通常会调用ejbRemove
- ❑ B. **onMessage()**方法可以抛出应用异常
- ❑ C. 消息驱动bean只能使用CMT划分
- ❑ D. **getRollbackOnly()**方法只能从onMessage方法调用

**14** 关于消息驱动bean类，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 必须实现（直接或间接）**javax.jms.Message**
- ❑ B. 必须有一个公共构造函数，取一个类型为**javax.jms.Message**的参数
- ❑ C. 允许实现 **finalize()**方法
- ❑ D. 实现**ejbCreate()**方法是可选的
- ❑ E. 这个类不能声明为"final"

**15** 在部署描述文件的**<ejb-local-ref>**元素中，哪些元素是可选的？（选出所有正确的答案。）

- ❑ A. **<local>**
- ❑ B. **<ejb-link>**
- ❑ C. **<local-home>**
- ❑ D. **<description>**
- ❑ E. **<ejb-ref-name>**
- ❑ F. **<ejb-ref-type>**

**16** 关于资源管理器连接工厂，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. **<res-sharing-scope>**部署描述文件元素用于指示一个资源管理器的连接是否可以在应用中的多个EJB之间共享
- ❑ B. **<res-sharing-scope>**部署描述文件元素包含**<resource-ref>**元素
- ❑ C. bean的所有资源管理器连接工厂引用都声明在一个**<resource-ref>**元素中，使用**<res-ref-name>**元素来加以区别
- ❑ D. 默认地，一个给定资源管理器的连接可以在应用中的多个bean之间共享

**17** 关于EJB代码中引用的安全角色，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 在部署描述文件中，这些引用包含在**<security-role-ref>**元素中
- ❑ B. **<security-role>**元素包括**<security-role-ref>**元素
- ❑ C. 在 **<security-role-ref>**元素中，**<role-name>**元素的值与bean的isCallerInRole方法调用的参数相同
- ❑ D. **<role-link>**元素用于链接两个**<security-role-ref>**元素

**18** 关于客户的安全视图，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 事务性客户不能在一个事务中改变它的主体关联
- ❑ B. 会话bean的客户在与会话对象通信期间不能改变它的主体关联
- ❑ C. 一个事务中的事务性请求不可能来自多个客户
- ❑ D. 以上都不对

**19** 编写一个实体bean类时，如果想确保这个bean在所有EJB 2.0容器上都可移植，应当避免哪些技术？（选出所有正确的答案。）

- ❑ A. 改变线程的优先级
- ❑ B. 使用反射API
- ❑ C. 使用包装器类
- ❑ D. 使用静态嵌套类

**20** 编写一个消息驱动bean类时，如果想确保这个bean在所有EJB 2.0容器上都可移植，应当避免哪些技术？（选出所有正确的答案。）

- ❑ A. 使用GUI的Swing API
- ❑ B. 使用“transient”修饰符
- ❑ C. 使用本地库
- ❑ D. 读取文件描述符

**21** 容器钝化一个有状态会话bean时，对容器有哪些要求？

- ❑ A. bean的实例状态总是要经过Java编程语言的串行化（Serialization）
- ❑ B. 必须把bean的所有实例字段状态保存起来，而不论字段的修饰符是什么
- ❑ C. 必须保存bean的 SessionContext对象的引用
- ❑ D. 必须保存所有非空的临时（transient）变量

**22** `javax.ejb.EJBLocalObject` 接口中定义了哪些功能？（选出所有正确的答案。）

- ❑ A. 删除一个实体对象
- ❑ B. 得到一个实体对象的主键
- ❑ C. 得到实体对象的一个本地home接口
- ❑ D. 得到实体对象的一个远程组件接口
- ❑ E. 向客户展现`javax.ejb.EntityBean` 接口的方法

**23** 关于ejbSelect方法，哪些说法是正确的？（选出所有正确的答案。）

❑ A. 可以展现给客户

❑ B. 只能返回EJBObject或EJBLocalObjects

❑ C. 只能由处于准备状态的bean调用

❑ D. 必须与部署描述文件中的一个查询元素关联

**24** 哪些方法绝对不会由消息驱动bean成功调用？（选出所有正确的答案。）

❑ A. **`isCallerInRole()`**

❑ B. **`getEJBHome()`**

❑ C. **`getRollbackOnly()`**

❑ D. **`setRollbackOnly()`**

❑ E. **`getCallerPrincipal()`**

**25** 通常由哪些角色负责向EJB的部署描述文件增加<ejb-link>元素？

❑ A. bean提供者

❑ B. 应用组装人员

❑ C. 部署人员

❑ D. 容器提供者

❑ E. 系统管理员

**26** 无状态会话bean可以在以下哪个方法中调用**`isCallerInRole()`**方法来完成一个安全检查？（选出所有正确的答案。）

❑ A. **`ejbCreate()`**

❑ B. **`ejbRemove()`**

❑ C. **`setSessionContext()`**

❑ D. 以上都不对

**27** 通常由哪两个角色负责创建ejb-jar文件？（有两个答案）

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 系统管理员

**28** 以下哪个无状态会话bean容器回调方法要取一个参数？（选出所有正确的答案。）

- ❑ A. **ejbRemove**
- ❑ B. **ejbCreate**
- ❑ C. **ejbCreate**
- ❑ D. **ejbPassivate**
- ❑ E. **setSessionContext**

**29** 以下哪些方法无法在SessionContext接口上调用？ （选出所有正确的答案。）

- ❑ A. **getEJBHome()**
- ❑ B. **getEJBObject()**
- ❑ C. **getEJBTransaction()**
- ❑ D. **isCallerInRole()**
- ❑ E. **getUserTransaction()**

**30** 关于实体bean的远程组件接口，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 如果客户试图在一个不存在的实体上调用一个方法，会抛出一个**java.rmi.NoSuchRemoteObjectException**异常。
- ❑ B. 必须扩展**javax.ejb.EJBHome**
- ❑ C. 其方法必须声明**java.rmi.RemoteException**
- ❑ D. 定义了一个**remove(Handle handle)**方法。

**31** 关于一个实体bean的本地home接口中的查找方法，哪些说法是正确的？（选出所有正确的答案。）

❑ A. 可以取任何合法的Java名

❑ B. 必须都声明 **javax.ejb.FinderException**

❑ C. 可以（可选）声明**java.rmi.RemoteException**

❑ D. **findByPrimaryKey**方法可以重载

❑ E. **findByPrimaryKey**方法的返回类型必须是bean的本地组件接口

❑ F. 名为"findXXX"的方法必须以bean的本地组件接口作为其声明的返回类型

---

**32** 关于消息驱动bean，哪些说法是正确的？（选出所有正确的答案。）

❑ A. 部署人员使用部署描述文件来确定一个bean是面向队列（Queue）还是面向主题（Topic）

❑ B. 这个类必须声明为final

❑ C. 这个类必须声明一个**ejbRemove()**方法

❑ D. 这个类可以有重载的**ejbCreate()**方法

❑ E. 这个类必须定义一个无参数的**onMessage**方法

---

**33** 以下是部署描述文件中的一段：

```
<entity>
  <ejb-name>Payroll</ejb-name>
  <security-role-ref>
      <role-name>clerk</role-name>
  </security-role-ref>
</entity>
```

下面哪些代码段可以完成一个合法的安全检查？（选出所有正确的答案。）

❑ A. **context.isCallerInRole( "Payroll" );**

❑ B. **context.isCallerInRole( "clerk" );**

❑ C. **context.isCallerInRole( "Payroll.clerk" );**

❑ D. **context.isCallerInRole( "Payroll/clerk" );**

**34** 以下哪个是有状态会话bean的合法的本地home接口？（选出所有正确的答案。）

- ❑ A.
```
public interface TestBean implements
javax.ejb.EJBLocalHome{
void create() throws CreateException;
}
```
- ❑ B.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal ejbCreate() throws CreateException;
}
```
- ❑ C.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal create() throws CreateException;
}
```
- ❑ D.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal create() throws CreateException,
RemoteException;
}
```
- ❑ E.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal create() throws LocalException;
}
```

**35** 哪些方法可以由bean提供者调用？（选出所有正确的答案。）

- ❑ A. `remove()`
- ❑ B. `ejbCreate()`
- ❑ C. `afterBegin()`
- ❑ D. `getCallerPrincipal()`
- ❑ E. `ejbPassivate()`

**36** 通常由谁在bean的部署描述文件中指定一个bean是否再入？

- ❑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 系统管理员

**37** 关于有状态会话bean和无状态会话bean，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 仅有状态会话bean支持事务
- ❑ B. 仅有状态会话bean可以钝化
- ❑ C. 仅有状态会话bean有一个“**setSessionContext**”方法
- ❑ D. 有状态会话bean和无状态会话bean都可以支持重载的“**ejbCreate**”方法
- ❑ E. 有状态会话bean和无状态会话bean都可以实现**javax.ejb.SessionSynchronization**接口
- ❑ F. 有状态会话bean和无状态会话bean都可以有实例变量状态

**38** 每个ejb-jar文件中必须包括什么？（选出所有正确的答案。）

- ❑ A. EJBHome接口的桩，可以直接包括，也可以通过引用
- ❑ B. JAR清单文件
- ❑ C. 一个部署描述文件
- ❑ D. JNDI上下文
- ❑ E. EJB的home接口，可以直接包括，也可以通过引用

**39** 哪些方法在javax.ejb.EJBHome接口中声明？（选出所有正确的答案。）

- ❑ A. **create()**
- ❑ B. **lookup()**
- ❑ C. **getHandle()**
- ❑ D. **getHomeHandle()**
- ❑ E. **setSessionContext()**

**40** 从一个会话bean客户可以无异常地调用多少个javax.ejb.EJBLocalHome接口方法?

- ❑ A. 0
- ❑ B. 1
- ❑ C. 2
- ❑ D. 3
- ❑ E. 4

**41** 以下哪个方法若调用，总是与会话**bean**的业务方法有相同的事务上下文？（选出所有正确的答案。）

- ❑ A. 构造函数
- ❑ B. **setSessionContext()**
- ❑ C. **afterBegin()**
- ❑ D. **afterCompletion()**
- ❑ E. **beforeCompletion()**

**42** 给定一个使用容器托管事务划分的有状态会话bean，从哪个方法可以访问另一个bean？（选出所有正确的答案。）

- ❑ A. **setSessionContext()**
- ❑ B. **ejbCreate()**
- ❑ C. **afterBegin()**
- ❑ D. **beforeCompletion()**
- ❑ E. **afterCompletion()**

**43** 哪些部署描述文件元素可以可选地指定级联删除功能？（选出所有正确的答案。）

- ❑ A. **<ejb-relation>**
- ❑ B. **<abstract-schema-name>**
- ❑ C. **<ejb-relationship-role>**
- ❑ D. **<relationship-role-source>**

**44** 关于级联删除，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 要在`ejbRemove()`方法中声明
- ❑ B. 如果一个支持级联删除的bean被删除，会导致与它有关系的所有bean都自动删除
- ❑ C. 可以为一对一、一对多或多对多关系指定级联删除
- ❑ D. 对于一个支持级联删除的bean，如果与它有关系的任何“多重性为1”的bean被删除，这个bean也会自动被删除

**45** 不论事务上下文是什么，哪些方法可以从EntityContext接口调用？（选出所有正确的答案。）

- ❑ A. `getEJBHome()`
- ❑ B. `getEJBObject()`
- ❑ C. `setRollbackOnly()`
- ❑ D. `getRollbackOnly()`

**46** 关于实体bean的主键，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 主键字段必须是cmp-fields
- ❑ B. 与主键相关的字段的设置方法不能通过客户视图对客户展现
- ❑ C. 如果两个实体对象有相同的home和相同的主键，则认为它们是相等的
- ❑ D. `getPrimaryKey()`只能在远程home接口的引用上调用，而不能在本地home接口的引用上调用

**47** bean提供者使用哪些方法来完成BMT划分？（选出所有正确的答案。）

- ❑ A. `javax.transaction.Transaction`
- ❑ B. `javax.transaction.UserTransaction`
- ❑ C. `javax.transaction.Synchronization`
- ❑ D. `javax.transaction.TransactionManager`

**48** 以下哪个是合法的EJB QL查询？（选出所有正确的答案。）

- ❑ A. SELECT c
  FROM Customer c
- ❑ B. SELECT (OBJECT) c.name
  FROM Customer c
- ❑ C. SELECT OBJECT c
  WHERE Customer c
- ❑ D. SELECT OBJECT (c)
  FROM Customer c
- ❑ E. SELECT c.name
  FROM Customer c

---

**49** 关于EJB 2.0中的事务，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 在一个事务中只能更新一个数据库
- ❑ B. 使用BMT划分的实体bean必须使用setStatus方法而不是**`setRollbackOnly`**方法
- ❑ C. 使用BMT划分的实体bean必须使用getStatus方法而不是**`getRollbackOnly`** 方法
- ❑ D. bean访问不支持事务的资源管理器时应当使用BMT划分
- ❑ E. “SessionSynchronization”接口只能由有状态会话bean使用

---

**50** 为了确保bean可移植性，对于一个使用CMP的实体bean，应当在其组件接口的业务方法上使用什么事务属性？（选出所有正确的答案。）

- ❑ A. NotSupported
- ❑ B. Required
- ❑ C. Supports
- ❑ D. RequiresNew
- ❑ E. Mandatory
- ❑ F. Never

**51** 以下哪个异常会导致一个bean被容器删除？ （选出所有正确的答案。）

- ❑ A. **javax.ejb.ObjectNotFoundException**
- ❑ B. **javax.ejb.CreateException**
- ❑ C. **javax.ejb.NoSuchEntityException**
- ❑ D. **javax.ejb.FinderException**
- ❑ E. **javax.ejb.RemoveException**

**52** 对于使用CMT划分的bean，在什么情况下，容器会把这样一个bean的业务方法所抛出的一个异常记入日志？ （选出所有正确的答案。）

- ❑ A. 如果方法在一个未指定的事务上下文中运行，并抛出一个系统异常
- ❑ B. 如果方法在一个未指定的事务上下文中运行，并抛出一个应用异常
- ❑ C. 如果方法在一个RequiresNew上下文中运行，并抛出一个系统异常
- ❑ D. 如果方法在一个Required上下文中运行，并抛出一个系统异常
- ❑ E. 如果方法在一个RequiresNew上下文中运行，并抛出一个应用异常
- ❑ F. 如果方法在一个Required上下文中运行，并抛出一个应用异常

**53** 会话bean组件接口的哪些方法可以由一个本地客户成功地调用？ （选出所有正确的答案。）

- ❑ A. **remove()**
- ❑ B. **getHandle()**
- ❑ C. **isIdentical()**
- ❑ D. **getEJBHome()**

**54** 在什么情况下，bean的业务方法可以调用SessionContext接口的**isCallerInRole()**方法？（选出所有正确的答案。）

❑ A. 使用容器托管事务划分的有状态会话bean

❑ B. 使用容器托管事务划分的无状态会话bean

❑ C. 使用bean托管事务划分的有状态会话bean

❑ D. 使用bean托管事务划分的无状态会话bean

❑ E. 以上都不对

---

**55** 创建一个使用容器托管持久存储的实体bean时，以下哪个可以通过这个bean的远程组件接口访问？（选出所有正确的答案。）

❑ A. 关系字段的存取方法

❑ B. 实体bean的本地接口

❑ C. bean的业务方法

❑ D. 用于容器托管关系的集合类

❑ E. 持久字段的存取方法

---

**56** 给定一个容器托管的一对一双向关系：

```
Foo  <--->  Bar
```

以及对象关系：

```
f1  <--->  b1
f2  <--->  b2
```

运行以下代码之后，哪个布尔表达式为真？

```
b2.setFoo(b1.getFoo());
```

（选出所有正确的答案。）

❑ A. **f1.getBar() == null**

❑ B. **b2.getFoo() == null**

❑ C. **f2.getBar() == null**

❑ D. 以上都不是

**57** 关于实体bean的相等性，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 如果两个实体对象引用用==操作符比较，容器提供者不一定生成一致的结果
- ❑ B. 如果两个实体对象引用使用**`equals()`**方法比较，若这两个实体对象有相同的主键，容器会返回true
- ❑ C. 如果两个实体对象引用使用**`isIdentical()`**方法比较，若这两个实体对象有相同的主键，容器会返回true
- ❑ D. 以上说法都不对

---

**58** 给定CMP bean：CustomerBean、OrderBean和LineItemsBean，它们有以下关系：

CustomerBean (1) <----> OrderBean (n)

OrderBean (1) <----> LineItemsBean (n)

以下哪个查询会返回一个有订单顾客的集合？（选出所有正确的答案。）

- ❑ A. SELECT Customer<br>FROM Order
- ❑ B. SELECT DISTINCT Customer<br>FROM Order
- ❑ C. SELECT o.custnum<br>FROM Order o, Customer c<br>WHERE c.custnum = o.custnum
- ❑ D. SELECT OBJECT (c)<br>FROM Customer c, Order o<br>WHERE c.custnum = o.custnum
- ❑ E. SELECT DISTINCT OBJECT (c)<br>FROM Customer c, Order o<br>WHERE c.custnum = o.custnum

**59** 给定：

```
Method-1 (Tx-1) -->  Method-2 (Tx-1) ---> Method-3 (Tx-1)
                \-->  Method-4 (Tx-2)
```

如果箭头左边的是调用方法，箭头右边的是被调用方法，哪组事务属性会支持指定的事务作用域？（选出所有正确的答案。）

- ❑ A. **M1 = Never**
  **M2 = Supports**
  **M3 = Required**
  **M4 = RequiresNew**
- ❑ B. **M1 = Required**
  **M2 = Supports**
  **M3 = Mandatory**
  **M4 = RequiresNew**
- ❑ C. **M1 = RequiresNew**
  **M2 = RequiresNew**
  **M3 = Supports**
  **M4 = RequiresNew**
- ❑ D. **M1 = Required**
  **M2 = Mandatory**
  **M3 = Supports**
  **M4 = Supports**

**60** 给定以下EJB QL表达式：

**p.discount NOT BETWEEN 10 AND 15**

它与哪个表达式等价？

- ❑ A. **p.discount < 10 OR p.discount > 15**
- ❑ B. **p.discount <= 10 OR p.discount > 15**
- ❑ C. **p.discount < 10 OR p.discount >= 15**
- ❑ D. **p.discount <= 10 OR p.discount >= 15**

**61** 当实体bean中的一个业务方法调用`getRollbackOnly()`方法时，哪个事务属性设置会导致容器抛出一个异常？（选出所有正确的答案。）

- ❑ A. NotSupported
- ❑ B. Required
- ❑ C. Supports
- ❑ D. RequiresNew
- ❑ E. Mandatory
- ❑ F. Never

**62** 如果一个消息驱动bean在onMessage()方法结束前没有完成它的事务，容器会做以下的什么事情？（选出所有正确的答案。）

- ❑ A. 将一个应用错误记入日志
- ❑ B. 回滚所开始的事务
- ❑ C. 删除bean实例
- ❑ D. 抛出一个异常

**63** 应用异常可以扩展哪个类？（选出所有正确的答案。）

- ❑ A. `java.lang.Exception`
- ❑ B. `java.lang.RuntimeException`
- ❑ C. `java.rmi.RemoteException`
- ❑ D. `javax.ejb.CreateException`

**64** 关于应用异常，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 应用异常要由系统管理员处理
- ❑ B. 应用异常应当报告系统级问题
- ❑ C. 会导致自动将事务标记为回滚
- ❑ D. 不能扩展`java.lang.RuntimeException`
- ❑ E. 可以扩展`java.rmi.RemoteException`

**65** 在什么情况下，就算容器没有调用**ejbRemove()**，也可能导致一个会话bean实例被撤销？（选出所有正确的答案。）

- ❑ A. 在一个使用容器托管事务划分的有状态会话bean上回滚一个事务
- ❑ B. 从业务方法抛出了一个系统异常
- ❑ C. bean处于钝化状态时，一个不活动的客户超时
- ❑ D. bean在一个事务中时客户调用了删除方法

**66** 哪些部署描述文件元素用于指定一个**Collection**接口？（选出所有正确的答案。）

- ❑ A. **<ejb-name>**
- ❑ B. **<cmr-field>**
- ❑ C. **<ejb-relation>**
- ❑ D. **<cmr-field-type>**
- ❑ E. **<abstract-schema-name>**

**67** 如果一个bean捕获了一个受查异常，而且无法从中恢复，它该做什么？（选出所有正确的答案。）

- ❑ A. 如果客户是远程客户，则抛出一个**java.rmi.RemoteException**
- ❑ B. 打印一个栈轨迹
- ❑ C. 不论客户是本地还是远程，都抛出一个**javax.ejb.EJBException**
- ❑ D. 不论客户是本地还是远程，把这个异常再原样传播给容器

**68** 对CMP实体bean类有哪些要求？（选出所有正确的答案。）

- ❑ A. 所有**ejbSelect()**方法都必须声明为抽象方法（abstract）
- ❑ B. 这个类不能定义**ejbCreate**方法
- ❑ C. **ejbPostCreate**方法由容器实现
- ❑ D. 辅助方法不能在bean类中实现
- ❑ E. 不能实现以“**ejbFind**”开头的方法

**69** 以下哪个部署描述文件元素是合法的，可以定义一个cmr-field？（选出所有正确的答案。）

❑ A.
```
<ejb-relationship-role>
<!-- assume mandatory ejb-relationship-role elements
inserted here -->
   <cmr-field>
       <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
</ejb-relationship-role>
```

❑ B.
```
<ejb-relationship-role>
<!-- assume mandatory ejb-relationship-role elements
inserted here -->
   <cmr-field>
       <cmr-field-name>lineItems</cmr-field-name>
       <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
<ejb-relationship-role>
```

❑ C.
```
<relationship-role-source>
<!-- assume mandatory relationship-role-source
elements inserted here -->
   <cmr-field>
       <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
</relationship-role-source>
```

❑ D.
```
<relationship-role-source>
<!-- assume mandatory relationship-role-source
elements inserted here -->
   <cmr-field>
       <cmr-field-name>lineItems</cmr-field-name>
       <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
</relationship-role-source>
```

**70** 如果远程客户接收到实体bean的异常，关于这些异常的远程客户视图，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 如果容器把一个事务标记为回滚，容器总会发出一个**javax.transaction.TransactionRolledbackException**异常
- ❑ B. 如果客户接收到一个**javax.transaction.TransactionRolledBackException**，容器会保证这个事务绝对不会提交
- ❑ C. **javax.transaction.TransactionRequiredException**异常会通知bean要在一个事务上下文中调用
- ❑ D. 如果容器发现所要的一个bean不复存在，它总会抛出**javax.ejb.NoSuchObjectEntityException**

# COFFEE CRAM

# 最后的模拟测验答案

**1** EJB 2.0有什么好处？（选出所有正确的答案。） (规范：108)

- [ ] A．服务器瘫痪后，MDB还可幸存
- [x] B．一个实体的表示可以由多个客户共享
- [ ] C．支持嵌套事务
- [ ] D．动态服务恢复 — 这是Jini的功能
- [ ] E．可以用声明方式设置隔离级别

**2** 哪些方法是客户直接调用的？ （选出所有正确的答案。）

- [ ] A. `ejbPassivate()` — 由容器调用
- [x] B. 业务方法
- [ ] C. `setSessionContext()`
- [ ] D. `newInstance()` — 由容器调用
- [x] E. `create()`

**3** EJBHome接口里有哪些方法？（选出所有正确的答案。） (API文档)

- [x] A. remove(Handle handle)
- [x] B. remove(Object primaryKey)
- [ ] C. getHandle() — 这是EJBObject中的方法
- [ ] D. 以上都不是

**4** 哪些方法可以由处于入池状态的CMP bean运行？（选出所有正确的答案。） （规范：168）

- ❑ A. **ejbLoad()**
- ☑ B. **ejbFind()** — bean不会因为home方法和查找方法离开池
- ❑ C. **ejbStore()**
- ❑ D. **ejbCreate()**
- ❑ E. 业务方法
- ☑ F. **ejbHome()**

**5** 关于消息驱动bean，哪些说法是正确的？（选出所有正确的答案。） （规范：314，316，324）

- ☑ A. 对消息驱动bean实例的所有调用都必须串行化
- ❑ B. 容器能保证消息会按其接收的顺序得到处理
- ❑ C. bean的 **ejbCreate()** 方法必须取一个类型为 **javax.jms.Message**的参数 — 必须有一个无参数的创建方法，onMessage方法才取消息作为参数
- ☑ D. bean提供者使用部署描述文件来指示bean类的实例面向主题还是队列

**6** 关于一个企业bean的环境，哪些说法是正确的？ （选出所有正确的答案。） （规范：412～413）

- ❑ A. 在bean能访问其环境项之前，这个bean必须先使用一个**SessionContext**对象得到命名上下文 — 不要把JNDI上下文与EJBContext（如SessionContext）搞混了
- ❑ B. 只有bean提供者才能设置环境项值 — 通常由部署人员或应用组装人员设置
- ☑ C. bean的环境项只能保存在“**java:comp/env**” 或其某个子上下文中
- ☑ D. 对于bean代码中的每个环境项查找，在bean的部署描述文件中都必须有一个匹配的**<env-entry>** 元素 — 否则，代码就没有东西可查了！

**7** 关于EJB中的安全角色，哪些说法是正确的？（选出所有正确的答案。） （规范：440～443）

- ☑ A. 安全角色是使用<security-role>元素在部署描述文件中定义的
- ❑ B. 安全角色的作用范围是实例级的
- ☑ C. 多个方法可以映射到一个安全角色
- ❑ D. 一个方法只能出现在一个安全角色中

**8** 一个EJB 2.0容器必须至少支持哪些API的一个子集？（选出所有正确的答案。）（规范：493～493）

- ☑ A. JTA 1.0.1
- ☑ B. JDBC 2.0
- ☑ C. JMS 1.0.2
- ☐ D. JAX-RCP 1.0

**9** 关于无状态会话bean，哪些说法是正确的？（选出所有正确的答案。）？（规范：95）

- ☑ A. 可以使用bean托管事务划分
- ☑ B. 必须有一个无参数创建方法
- ☐ C. 一个实例可以支持并发调用
- ☐ D. 必须扩展javax.ejb.SessionBean — 要实现（implements）才对 :)

**10** 哪个功能只在以下的某一个接口中有（而不是两个接口中都有）

–实体对象的远程组件接口

–实体对象的本地组件接口

（选出所有正确的答案。）

- ☐ A. 删除对象
- ☑ B. 得到对象的句柄 本地接口没有
- ☐ C. 调用业务方法
- ☐ D. 得到对象的主键

**11** 创建一个CMP实体bean时，哪些方法是可选的？（选出所有正确的答案。）（规范：171～174）

- ☐ A. `ejbLoad()`
- ☑ B. `ejbCreate()` — 不必允许客户插入新实体
- ☐ C. `ejbRemove()`
- ☑ D. `ejbSelect()`
- ☐ E. `ejbPassivate()`
- ☐ F. `setEntityContext()`

**12** 哪些方法可以在处于入池状态的bean上调用？（选出所有正确的答案。） （规范：168）

- ☑ A. `ejbFind()`
- ☐ B. `ejbLoad()`
- ☐ C. `ejbStore()`
- ☑ D. `ejbSelect()`
- ☐ E. `ejbPassivate()`

查找方法和选择方法并非特定于一个实体

**13** 关于消息驱动bean的生命周期，哪些说法是正确的？（选出所有正确的答案。） （规范：318～320）

- ☐ A. `onMessage`方法完成时，容器通常会调用ejbRemove — bean只是回到池中
- ☐ B. `onMessage()`方法可以抛出应用异常 — 抛给谁呢？
- ☐ C. 消息驱动bean只能使用CMT划分
- ☑ D. `getRollbackOnly()`方法只能从onMessage方法调用

**14** 关于消息驱动bean类，哪些说法是正确的？（选出所有正确的答案。） （规范：323）

- ☐ A. 必须实现（直接或间接）`javax.jms.Message` — 应该是MessageListener
- ☐ B. 必须有一个公共构造函数，取一个类型为`javax.jms.Message`的参数
- ☐ C. 允许实现`finalize()`方法 —不行！任何bean中都不行
- ☐ D. 实现`ejbCreate()`方法是可选的
- ☑ E. 这个类不能声明为"final" — 容器可能希望部署你的bean的一个子类

**15** 在部署描述文件的`<ejb-local-ref>`元素中，哪些元素是可选的？（选出所有正确的答案。） （规范：466）

- ☐ A. `<local>`
- ☑ B. `<ejb-link>`
- ☐ C. `<local-home>`
- ☑ D. `<description>`
- ☐ E. `<ejb-ref-name>`
- ☐ F. `<ejb-ref-type>`

（规范：423，481）

**16** 关于资源管理器连接工厂，哪些说法是正确的？（选出所有正确的答案。）

☑ A. **<res-sharing-scope>**部署描述文件元素用于指示一个资源管理器的连接是否可以在应用中的多个EJB之间共享

❑ B. **<res-sharing-scope>**部署描述文件元素包含**<resource-ref>**元素 — 反了

❑ C. bean的所有资源管理器连接工厂引用都声明在一个**<resource-ref>**元素中，使用**<res-ref-name>**元素来加以区别 — 每个引用都必须用一个<resource-ref>单独指定

☑ D. 默认地，一个给定资源管理器的连接可以在应用中的多个bean之间共享

（规范：439～441，446）

**17** 关于EJB代码中引用的安全角色，哪些说法是正确的？（选出所有正确的答案。）

☑ A. 在部署描述文件中，这些引用包含在**<security-role-ref>**元素中

❑ B. **<security-role>**元素包括**<security-role-ref>**元素

☑ C. 在**<security-role-ref>**元素中，**<role-name>**元素的值与bean的isCallerInRole方法调用的参数相同 — 这与实际的<security-role>角色名不同

❑ D. **<role-link>**元素用于链接两个**<security-role-ref>**元素 — 不对，它把一个<security-role-ref>链接到一个<security-role>

（规范：450）

**18** 关于客户的安全视图，哪些说法是正确的？（选出所有正确的答案。）

☑ A. 事务性客户不能在一个事务中改变它的主体关联

☑ B. 会话bean的客户在与会话对象通信期间不能改变它的主体关联

❑ C. 一个事务中的事务性请求不可能来自多个客户

❑ D. 以上都不对

**19** 编写一个实体bean类时，如果想确保这个bean在所有EJB 2.0容器上都可移植，应当避免哪些技术？（选出所有正确的答案。） （规范：494～495）

- ☑ A. 改变线程的优先级
- ☑ B. 使用反射API
- ☐ C. 使用包装器类
- ☐ D. 使用静态嵌套类

**20** 编写一个消息驱动bean类时，如果想确保这个bean在所有EJB 2.0容器上都可移植，应当避免哪些技术？（选出所有正确的答案。） （规范：494～495）

- ☑ A. 使用GUI的Swing API
- ☐ B. 使用“transient”修饰符
- ☑ C. 使用本地库
- ☑ D. 读取文件描述符

**21** 容器钝化一个有状态会话bean时，对容器有哪些要求？ （规范：71）

- ☐ A. bean的实例状态总是要经过Java编程语言的串行化（Serialization） — 可以是类似于串行化的一种钝化过程
- ☐ B. 必须把bean的所有实例字段状态保存起来，而不论字段的修饰符是什么
- ☑ C. 必须保存bean的 SessionContext对象的引用
- ☐ D. 必须保存所有非空的临时（transient）变量 — 不对，并不保存临时变量，但是可能不会重新把它们设置为默认值……

**22** `javax.ejb.EJBLocalObject` 接口中定义了哪些功能？（选出所有正确的答案。）

- ☑ A. 删除一个实体对象
- ☑ B. 得到一个实体对象的主键
- ☑ C. 得到实体对象的一个本地home接口
- ☐ D. 得到实体对象的一个远程组件接口
- ☐ E. 向客户展现`javax.ejb.EntityBean` 接口的方法

**23** 关于ejbSelect方法，哪些说法是正确的？（选出所有正确的答案。） （规范：183）

- ☐ A. 可以展现给客户
- ☐ B. 只能返回EJBObject或EJBLocalObjects — 几乎可以返回任何东西
- ☐ C. 只能由处于准备状态的bean调用 — 不，可以由home业务方法调用。
- ☑ D. 必须与部署描述文件中的一个查询元素关联

**24** 哪些方法绝对不会由消息驱动bean成功调用？（选出所有正确的答案。） （规范：320～321）

- ☑ A. `isCallerInRole()`
- ☑ B. `getEJBHome()`
- ☐ C. `getRollbackOnly()`
- ☐ D. `setRollbackOnly()`
- ☑ E. `getCallerPrincipal()`

MDB没有客户视图，所以没有home或客户安全信息

**25** 通常由哪些角色负责向EJB的部署描述文件增加<ejb-link>元素？ （规范：418）

- ☐ A. bean提供者
- ☑ B. 应用组装人员 — 将所指的一个bean链接到另一个bean
- ☐ C. 部署人员
- ☐ D. 容器提供者
- ☐ E. 系统管理员

**26** 无状态会话bean可以在以下哪个方法中调用`isCallerInRole()`方法来完成一个安全检查？（选出所有正确的答案。） （规范：90）

- ☐ A. `ejbCreate()`
- ☐ B. `ejbRemove()`
- ☐ C. `setSessionContext()`
- ☑ D. 以上都不对

— isCallerInRole()只能从业务方法调用

(规范：487)

**27** 通常由哪两个角色负责创建ejb-jar文件？（有两个答案）

- ☑ A. bean提供者
- ☑ B. 应用组装人员
- ☐ C. 部署人员
- ☐ D. 系统管理员

(规范：77)

**28** 以下哪个无状态会话bean容器回调方法要取一个参数？（选出所有正确的答案。）

- ☐ A. **ejbRemove**
- ☐ B. **ejbCreate**
- ☐ C. **ejbCreate**
- ☐ D. **ejbPassivate**
- ☑ E. **setSessionContext** — 取一个SessionContext对象作为参数

(规范：74)

**29** 以下哪些方法无法在SessionContext接口上调用？（选出所有正确的答案。）

- ☐ A. **getEJBHome()**
- ☐ B. **getEJBObject()**
- ☑ C. **getEJBTransaction()** — 没有这个方法:)
- ☐ D. **isCallerInRole()**
- ☐ E. **getUserTransaction()**

(规范：190)

**30** 关于实体bean的远程组件接口，哪些说法是正确的？（选出所有正确的答案。）

- ☐ A. 如果客户试图在一个不存在的实体上调用一个方法，会抛出一个**java.rmi.NoSuchRemoteObjectException**异常。 — 应该是 NoSuchObjectException
- ☐ B. 必须扩展**javax.ejb.EJBHome** — 应该是EJBObject
- ☑ C. 其方法必须声明**java.rmi.RemoteException**
- ☐ D. 定义了一个**remove(Handle handle)**方法。 — 只在home接口中才有

（规范：196）

**31** 关于一个实体bean的本地home接口中的查找方法，哪些说法是正确的？（选出所有正确的答案。）

❑ A. 可以取任何合法的Java名 — 必须以"find"开头

☑ B. 必须都声明 **javax.ejb.FinderException**

❑ C. 可以（可选）声明**java.rmi.RemoteException** — 本地接口中没有

❑ D. **findByPrimaryKey**方法可以重载

☑ E. **findByPrimaryKey**方法的返回类型必须是bean的本地组件接口

❑ F. 名为"findXXX"的方法必须以bean的本地组件接口作为其声明的返回类型 — 还可以是Collection

（规范：324~325，317）

**32** 关于消息驱动bean，哪些说法是正确的？（选出所有正确的答案。）

☑ A. 部署人员使用部署描述文件来确定一个bean是面向队列（Queue）还是面向主题（Topic）

❑ B. 这个类必须声明为final

☑ C. 这个类必须声明一个**ejbRemove()**方法

❑ D. 这个类可以有重载的**ejbCreate()**方法 — 没有客户

❑ E. 这个类必须定义一个无参数的**onMessage**方法 — onMessage要取一个消息作为参数

（规范：440）

**33** 以下是部署描述文件中的一段：

```
<entity>
  <ejb-name>Payroll</ejb-name>
  <security-role-ref>
      <role-name>clerk</role-name>
  </security-role-ref>
</entity>
```

下面哪些代码段可以完成一个合法的安全检查？（选出所有正确的答案。）

❑ A. **context.isCallerInRole("Payroll");**

☑ B. **context.isCallerInRole("clerk");**

❑ C. **context.isCallerInRole("Payroll.clerk");**

❑ D. **context.isCallerInRole("Payroll/clerk");**

**34** 以下哪个是有状态会话**bean**的合法的本地**home**接口？ （选出所有正确的答案。）（规范：98）

必须保证：
— 抛出CreateException
— 返回本地接口类型
— 以"create"开头
— 不能抛出RemoteException

❑ A.
```
public interface TestBean implements
javax.ejb.EJBLocalHome{
void create() throws CreateException;
}
```

❑ B.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal ejbCreate() throws CreateException;
}
```

☑ C.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal create() throws CreateException;
}
```

❑ D.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal create() throws CreateException,
RemoteException;
}
```

❑ E.
```
public interface TestBean extends
javax.ejb.EJBLocalHome {
TestBeanLocal create() throws LocalException;
}
```

**35** 哪些方法可以由bean提供者调用？ （选出所有正确的答案。）

☑ A. **remove()** — 回调方法是ejbRemove

❑ B. **ejbCreate()**

❑ C. **afterBegin()**

☑ D. **getCallerPrincipal()**

❑ E. **ejbPassivate()**

其他的是容器回调

(规范：457)

**36** 通常由谁在bean的部署描述文件中指定一个bean是否再人？

- ☑ A. bean提供者
- ❑ B. 应用组装人员
- ❑ C. 部署人员
- ❑ D. 系统管理员

**37** 关于有状态会话bean和无状态会话bean，哪些说法是正确的？（选出所有正确的答案。）

- ❑ A. 仅有状态会话bean支持事务
- ☑ B. 仅有状态会话bean可以钝化
- ❑ C. 仅有状态会话bean有一个“**setSessionContext**”方法
- ❑ D. 有状态会话bean和无状态会话bean都可以支持重载的“**ejbCreate**”方法 — 无状态会话bean只能一个无参数的ejbCreate
- ❑ E. 有状态会话bean和无状态会话bean都可以实现**javax.ejb.SessionSynchronization**接口 — 仅有状态会话bean才能实现这个接口。
- ☑ F. 有状态会话bean和无状态会话bean都可以有实例变量状态 — 无状态会话bean可以有状态，只不过不是客户特定的状态

(规范：488～490)

**38** 每个ejb-jar文件中必须包括什么？（选出所有正确的答案。）

- ❑ A. EJBHome接口的桩，可以直接包括，也可以通过引用 — 这是容器的任务
- ❑ B. JAR清单文件 — EJB 2.0对此没有要求
- ☑ C. 一个部署描述文件
- ❑ D. JNDI上下文
- ☑ E. EJB的home接口，可以直接包括，也可以通过引用

(API文档)

**39** 哪些方法在javax.ejb.EJBHome接口中声明？（选出所有正确的答案。）

- ❑ A. **create()**
- ❑ B. **lookup()**
- ❑ C. **getHandle()**
- ☑ D. **getHomeHandle()**
- ❑ E. **setSessionContext()**

**40** 从一个会话bean客户可以无异常地调用多少个javax.ejb.EJBLocalHome接口方法？ （规范：60）

- ☑ A. 0
- ☐ B. 1
- ☐ C. 2
- ☐ D. 3
- ☐ E. 4

EJBLocalHome中只有一个方法，就是取一个主键的删除方法

**41** 以下哪个方法若调用，总是与会话bean的业务方法有相同的事务上下文？（选出所有正确的答案。） （规范：76）

- ☐ A. 构造函数
- ☐ B. **setSessionContext()**
- ☑ C. **afterBegin()**
- ☐ D. **afterCompletion()** — 事务已经结束了
- ☑ E. **beforeCompletion()**

**42** 给定一个使用容器托管事务划分的有状态会话bean，从哪个方法可以访问另一个bean？（选出所有正确的答案。） （规范：80）

- ☐ A. **setSessionContext()**
- ☑ B. **ejbCreate()**
- ☑ C. **afterBegin()**
- ☑ D. **beforeCompletion()**
- ☐ E. **afterCompletion()**

setSessionContext和afterCompletion没有“有意义的事务上下文”

**43** 哪些部署描述文件元素可以可选地指定级联删除功能？（选出所有正确的答案。） （规范：468）

- ☐ A. **<ejb-relation>**
- ☐ B. **<abstract-schema-name>**
- ☑ C. **<ejb-relationship-role>**
- ☐ D. **<relationship-role-source>**

**44** 关于级联删除，哪些说法是正确的？（选出所有正确的答案。） （规范：133）

- ☐ A. 要在**ejbRemove()**方法中声明
- ☐ B. 如果一个支持级联删除的bean被删除，会导致与它有关系的所有bean都自动删除 — 级联删除是指"如果对方被删除，就把我也删除"
- ☐ C. 可以为一对一、一对多或多对多关系指定级联删除 — 不适用多对多关系
- ☑ D. 对于一个支持级联删除的bean，如果与它有关系的任何"多重性为1"的bean被删除，这个bean也会自动被删除

**45** 不论事务上下文是什么，哪些方法可以从EntityContext接口调用？（选出所有正确的答案。） （规范：180）

- ☑ A. **getEJBHome()**
- ☑ B. **getEJBObject()**
- ☐ C. **setRollbackOnly()**
- ☐ D. **getRollbackOnly()**

**46** 关于实体bean的主键，哪些说法是正确的？（选出所有正确的答案。） （规范：120，130）

- ☐ A. 主键字段必须是cmp-fields — 可以使用一个复合主键
- ☑ B. 与主键相关的字段的设置方法不能通过客户视图对客户展现
- ☑ C. 如果两个实体对象有相同的home和相同的主键，则认为它们是相等的
- ☐ D. **getPrimaryKey()**只能在远程home接口的引用上调用，而不能在本地home接口的引用上调用

**47** bean提供者使用哪些方法来完成BMT划分？（选出所有正确的答案。） （规范：332）

- ☐ A. **javax.transaction.Transaction**
- ☑ B. **javax.transaction.UserTransaction**
- ☐ C. **javax.transaction.Synchronization**
- ☐ D. **javax.transaction.TransactionManager**

(规范：236)

**48** 以下哪个是合法的EJB QL查询？（选出所有正确的答案。）

❏ A. SELECT c
FROM Customer c
— 需要 Object (c)或一个路径

❏ B. SELECT (OBJECT) c.name
FROM Customer c
— 全错了！必须是OBJECT(c)或c.name（见答案D）

❏ C. SELECT OBJECT c
WHERE Customer c
— 必须有一个FROM，还要有(c)

✔ D. SELECT OBJECT (c)
FROM Customer c

✔ E. SELECT c.name
FROM Customer c

(规范：339，348)

**49** 关于EJB 2.0中的事务，哪些说法是正确的？（选出所有正确的答案。）

❏ A. 在一个事务中只能更新一个数据库 — 这会妨害EJB 2.0中的一个重要特性，即分布式事务

❏ B. 使用BMT划分的实体bean必须使用setStatus方法而不是 **`setRollbackOnly`**方法

✔ C. 使用BMT划分的实体bean必须使用getStatus方法而不是 **`getRollbackOnly`** 方法

❏ D. bean访问不支持事务的资源管理器时应当使用BMT划分 — 对于"NotSupported"应使用CMT

✔ E. "SessionSynchronization"接口只能由有状态会话bean使用

(规范：352)

**50** 为了确保bean可移植性，对于一个使用CMP的实体bean，应当在其组件接口的业务方法上使用什么事务属性？（选出所有正确的答案。）

❏ A. NotSupported

✔ B. Required

❏ C. Supports

✔ D. RequiresNew

✔ E. Mandatory

❏ F. Never

**51** 以下哪个异常会导致一个bean被容器删除？（选出所有正确的答案。） （规范：372～374）

- ❑ A. **javax.ejb.ObjectNotFoundException**
- ❑ B. **javax.ejb.CreateException**
- ☑ C. **javax.ejb.NoSuchEntityException**
- ❑ D. **javax.ejb.FinderException**
- ❑ E. **javax.ejb.RemoveException**

其他都是可恢复的应用异常

**52** 对于使用CMT划分的bean，在什么情况下，容器会把这样一个bean的业务方法所抛出的一个异常记入日志？（选出所有正确的答案。） （规范：376）

- ☑ A. 如果方法在一个未指定的事务上下文中运行，并抛出一个系统异常
- ❑ B. 如果方法在一个未指定的事务上下文中运行，并抛出一个应用异常
- ☑ C. 如果方法在一个RequiresNew上下文中运行，并抛出一个系统异常
- ☑ D. 如果方法在一个Required上下文中运行，并抛出一个系统异常
- ❑ E. 如果方法在一个RequiresNew上下文中运行，并抛出一个应用异常
- ❑ F. 如果方法在一个Required上下文中运行，并抛出一个应用异常

— 系统异常总会记入日志

**53** 会话bean组件接口的哪些方法可以由一个本地客户成功地调用？（选出所有正确的答案。）

- ☑ A. **remove()**
- ❑ B. **getHandle()** — 本地客户视图没有句柄
- ☑ C. **isIdentical()**
- ❑ D. **getEJBHome()** — 应该是getEJBLocalHome()

**54** 在什么情况下，bean的业务方法可以调用SessionContext接口的`isCallerInRole()`方法？（选出所有正确的答案。）

- ☑ A. 使用容器托管事务划分的有状态会话bean　　在这里事务划分没有影响
- ☑ B. 使用容器托管事务划分的无状态会话bean
- ☑ C. 使用bean托管事务划分的有状态会话bean
- ☑ D. 使用bean托管事务划分的无状态会话bean
- ☐ E. 以上都不对

（规范：130）

**55** 创建一个使用容器托管持久存储的实体bean时，以下哪个可以通过这个bean的远程组件接口访问？（选出所有正确的答案。）

- ☐ A. 关系字段的存取方法
- ☐ B. 实体bean的本地接口
- ☑ C. bean的业务方法
- ☐ D. 用于容器托管关系的集合类
- ☑ E. 持久字段的存取方法　— 合法，但不好……

（规范：138）

**56** 给定一个容器托管的一对一双向关系：

```
Foo  <--->  Bar
```

以及对象关系：

```
f1  <--->  b1
f2  <--->  b2
```

运行以下代码之后，哪个布尔表达式为真？

```
b2.setFoo(b1.getFoo());
```

（选出所有正确的答案。）

- ☐ A. `f1.getBar() == null`
- ☐ B. `b2.getFoo() == null`
- ☑ C. `f2.getBar() == null`
- ☐ D. 以上都不是

b2取b1的Foo时，它会把它自己的foo2替换为foo1。所以，现在foo2没有bar。



**57** 关于实体bean的相等性，哪些说法是正确的？（选出所有正确的答案。） （规范：121）

☑ A. 如果两个实体对象引用用==操作符比较，容器提供者不一定生成一致的结果

☐ B. 如果两个实体对象引用使用**equals()**方法比较，若这两个实体对象有相同的主键，容器会返回true — 不能保证

☑ C. 如果两个实体对象引用使用**isIdentical()**方法比较，若这两个实体对象有相同的主键，容器会返回true

☐ D. 以上说法都不对

**58** 给定CMP bean：CustomerBean、OrderBean和LineItemsBean，它们有以下关系： （规范：232）

CustomerBean (1) <----> OrderBean (n)

OrderBean (1) <----> LineItemsBean (n)

以下哪个查询会返回一个有订单顾客的集合？（选出所有正确的答案。）

☐ A. SELECT Customer
FROM Order — 需要OBJECT或路径

☐ B. SELECT DISTINCT Customer
FROM Order — 需要OBJECT或路径

☐ C. SELECT o.custnum
FROM Order o, Customer c
WHERE c.custnum = o.custnum — 不是Customer类型

☐ D. SELECT OBJECT (c)
FROM Customer c, Order o
WHERE c.custnum = o.custnum — 不能保证避免重复

☑ E. SELECT DISTINCT OBJECT (c)
FROM Customer c, Order o
WHERE c.custnum = o.custnum

**59** 给定： （规范：357～359）

```
Method-1 (Tx-1) -->  Method-2 (Tx-1) ---> Method-3 (Tx-1)
                \-->  Method-4 (Tx-2)
```

如果箭头左边的是调用方法，箭头右边的是被调用方法，哪组事务属性会支持指定的事务作用域？ （选出所有正确的答案。）

- ❑ A. **M1 = Never**
  **M2 = Supports**
  **M3 = Required**
  **M4 = RequiresNew**
  — Never意味着……绝对没有。方法1的事务属性若是Never，那它绝对不会在一个事务中。
- ☑ B. **M1 = Required**
  **M2 = Supports**
  **M3 = Mandatory**
  **M4 = RequiresNew**
- ❑ C. **M1 = RequiresNew**
  **M2 = RequiresNew**
  **M3 = Supports**
  **M4 = RequiresNew**
  — 方法2在方法1的事务中运行，但是如果它的事务属性是RequiresNew，则要求是一个新事务。
- ❑ D. **M1 = Required**
  **M2 = Mandatory**
  **M3 = Supports**
  **M4 = Supports**
  — 方法4开始了一个新事务，所以显然不会以Supports作为事务属性。

**60** 给定以下EJB QL表达式： （规范：229）

**p.discount NOT BETWEEN 10 AND 15** — 10和15不包括在内

它与哪个表达式等价？

- ☑ A. **p.discount < 10 OR p.discount > 15**
- ❑ B. **p.discount <= 10 OR p.discount > 15**
- ❑ C. **p.discount < 10 OR p.discount >= 15**
- ❑ D. **p.discount <= 10 OR p.discount >= 15**

**61** 当实体bean中的一个业务方法调用`getRollbackOnly()`方法时，哪个事务属性设置会导致容器抛出一个异常？（选出所有正确的答案。） （规范：361）

- [x] A. NotSupported
- [ ] B. Required
- [x] C. Supports — 因为可能没有一个事务
- [ ] D. RequiresNew
- [ ] E. Mandatory
- [x] F. Never

**62** 如果一个消息驱动bean在onMessage()方法结束前没有完成它的事务，容器会做以下的什么事情？（选出所有正确的答案。） （规范：356）

- [x] A. 将一个应用错误记入日志
- [x] B. 回滚所开始的事务
- [x] C. 删除bean实例
- [ ] D. 抛出一个异常 — 抛给谁？

**63** 应用异常可以扩展哪个类？（选出所有正确的答案。） （规范：373）

- [x] A. `java.lang.Exception`
- [ ] B. `java.lang.RuntimeException` — 必须是受查异常。
- [ ] C. `java.rmi.RemoteException` — 绝对不能扩展这个类，尽管它确实是一个受查异常…… 这是bean法则规定的
- [x] D. `javax.ejb.CreateException`

**64** 关于应用异常，哪些说法是正确的？（选出所有正确的答案。） （规范：372–373）

- [ ] A. 应用异常要由系统管理员处理
- [ ] B. 应用异常应当报告系统级问题 要由客户处理
- [ ] C. 会导致自动将事务标记为回滚
- [x] D. 不能扩展`java.lang.RuntimeException`
- [ ] E. 可以扩展`java.rmi.RemoteException` — 不能扩展这个类

（规范：79）

**65** 在什么情况下，就算容器没有调用**ejbRemove()**，也可能导致一个会话bean实例被撤销？（选出所有正确的答案。）

- ☐ A. 在一个使用容器托管事务划分的有状态会话bean上回滚一个事务
- ☑ B. 从业务方法抛出了一个系统异常
- ☑ C. bean处于钝化状态时，一个不活动的客户超时
- ☐ D. bean在一个事务中时客户调用了删除方法 — 客户得到异常，但bean仍活着

（规范：462）

**66** 哪些部署描述文件元素用于指定一个Collection接口？（选出所有正确的答案。）

- ☐ A. **<ejb-name>**
- ☐ B. **<cmr-field>**
- ☐ C. **<ejb-relation>**
- ☑ D. **<cmr-field-type>** — 对于多重性为“多”的bean，用来指示Set或Collection
- ☐ E. **<abstract-schema-name>**

（规范：373）

**67** 如果一个bean捕获了一个受查异常，而且无法从中恢复，它该做什么？（选出所有正确的答案。）

- ☐ A. 如果客户是远程客户，则抛出一个**java.rmi.RemoteException** — bean不能抛出RemoteException。只有容器才能抛出这个异常。
- ☐ B. 打印一个栈轨迹
- ☑ C. 不论客户是本地还是远程，都抛出一个**javax.ejb.EJBException**
- ☐ D. 不论客户是本地还是远程，把这个异常再原样传播给容器

（规范：190–191）

**68** 对CMP实体bean类有哪些要求？（选出所有正确的答案。）

- ☑ A. 所有**ejbSelect()**方法都必须声明为抽象方法（abstract）
- ☐ B. 这个类不能定义**ejbCreate**方法
- ☐ C. **ejbPostCreate**方法由容器实现
- ☐ D. 辅助方法不能在bean类中实现
- ☑ E. 不能实现以“**ejbFind**”开头的方法

**69** 以下哪个部署描述文件元素是合法的，可以定义一个cmr-field？（选出所有正确的答案。） （规范：162，463）

❑ A.

```
<ejb-relationship-role>
<!-- assume mandatory ejb-relationship-role elements
inserted here -->
   <cmr-field>
      <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
</ejb-relationship-role>
```

—少了字段名

☑ B.

```
<ejb-relationship-role>
<!-- assume mandatory ejb-relationship-role elements
inserted here -->
   <cmr-field>
      <cmr-field-name>lineItems</cmr-field-name>
      <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
<ejb-relationship-role>
```

❑ C.

```
<relationship-role-source>
<!-- assume mandatory relationship-role-source
elements inserted here -->
   <cmr-field>
      <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
</relationship-role-source>
```

—少了名字，而且这里在角色源中，应该在角色中才对

❑ D.

```
<relationship-role-source>
<!-- assume mandatory relationship-role-source
elements inserted here -->
   <cmr-field>
      <cmr-field-name>lineItems</cmr-field-name>
      <cmr-field-type>java.util.Collection</cmr-field-type>
   </cmr-field>
</relationship-role-source>
```

—不能在角色源中

(规范：383)

**70** 如果远程客户接收到实体bean的异常，关于这些异常的远程客户视图，哪些说法是正确的？（选出所有正确的答案。）

- ☐ A. 如果容器把一个事务标记为回滚，容器总会发出一个**javax.transaction.TransactionRolledbackException**异常 —— 只有当调用者的事务回滚（而不是挂起）
- ☑ B. 如果客户接收到一个**javax.transaction.TransactionRolledBackException**，容器会保证这个事务绝对不会提交
- ☑ C. **javax.transaction.TransactionRequiredException**异常会通知bean要在一个事务上下文中调用 —— 如果事务属性是Mandatory就会发生这种情况。
- ☐ D. 如果容器发现所要的一个bean不复存在，它总会抛出**javax.ejb.NoSuchObjectEntityException** —— 应该是java.rmi.NoSuchObjectException



# 不是说再见

要用心地看看wickedlysmart.com

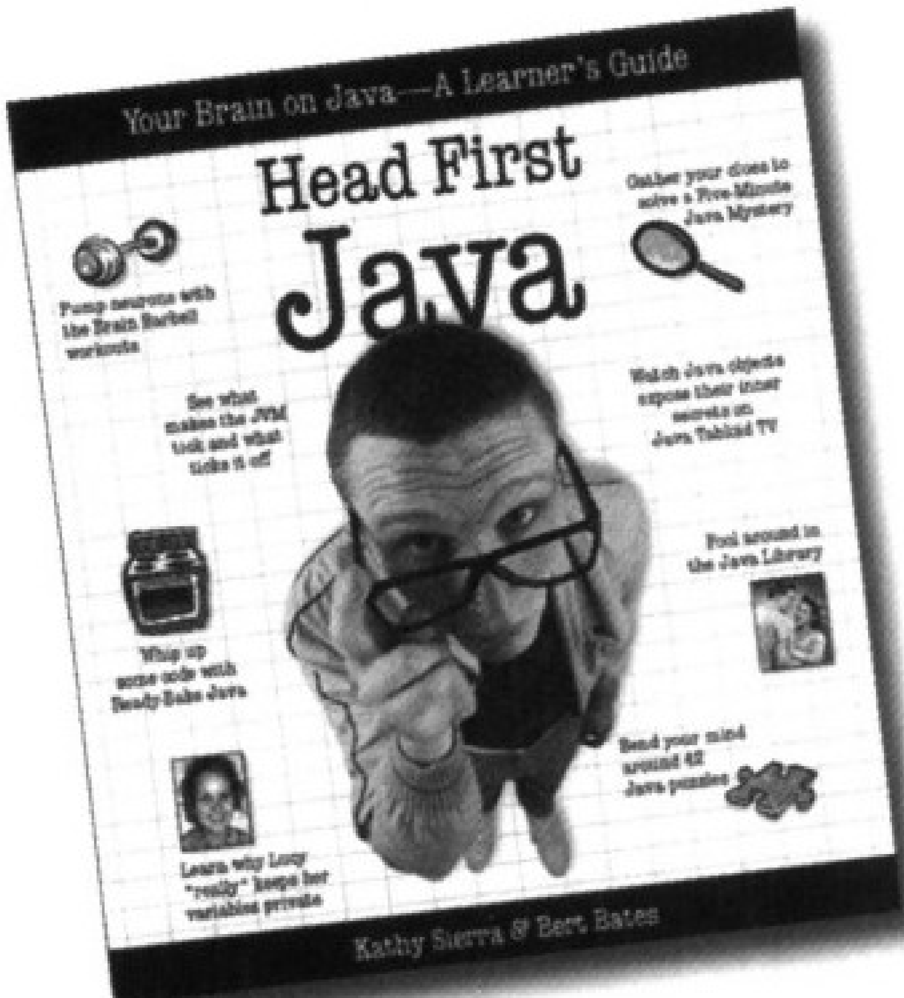
通过考试后别忘了写信告诉我们！

lkickedbutt@wickedlysmart.com
我们会为你喝上一杯。



# Head First系列的封面草案

做出这个决定很难，这是我们最后的定稿：




……我们费尽心思想了许多候选的封面……

1

我们觉得这样可以体现上手实践的思路……你觉得怎么样？ —K & B




既然要以头脑为先，为什么不把重点放在头（HEAD）上，把它叫做Head First呢？

-来自市场的声音

2

嘿，你想的"把重点放在头上"是不是这个样子？

—K & B

[嗯,不好 -来自市场的声音]




3

我们上周确定的这个封面怎么样？

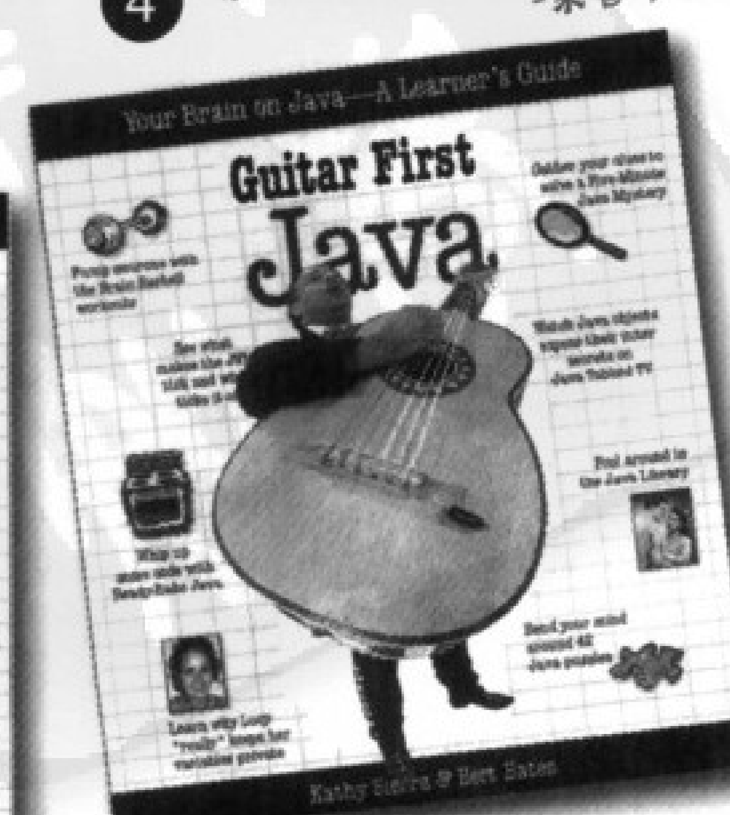
—K & B




4

我知道这一个更欢快，但是这到底代表什么呢？还是让专业人士来决定吧。

-来自市场的声音



# 索引

## 符号

<abstract-schema-name> 384, 394, 406
<acknowledge-mode> 460
<assembly-descriptor> 503
<cascade-delete/> 393
<cmp-field> 384
<cmr-field-name> 388
<cmr-field-type> 388
<cmr-field> 388
<container-transaction> 502
<destination-type> 449
<ejb-link> 615
<ejb-local-ref> 614
<ejb-ref-name> 613
<ejb-ref-type> 613
<ejb-ref> 613
<ejb-relation> 388
<ejb-relationship-role-name> 388
<ejb-relationship-role> 388
<env-entry-name> 607
<env-entry-type> 607
<env-entry-value> 607
<env-entry> 607
<exclude-list> 574
<message-driven-destination> 449
<message-driven> 449
<method-intf> 582
<method-param> 505
<method-params> 505, 579
<method-permission> 574, 578
<method> 504, 579
<multiplicity> 388, 422
<persistence-type> 385
<relationships> 395
<res-auth> 610, 611
<res-ref-name> 610
<res-sharing-scope> 610
<res-type> 610
<resource-env-ref> 616
<resource-ref> 610
<role-link> 589
<role-name> 574
<run-as> 591, 592
<security-identity> 591
<security-role-ref> 589
<security-role> 574
<trans-attribute> 476, 502, 504
<transaction-type> 476, 502
<unchecked/> 574, 581

# A

abstract（抽象） 313

abstract getters and setters（抽象获取方法和设置方法）315，383

abstract getter and setter（抽象获取方法和设置方法）313

abstract persistence schema（抽象持久存储模式）384，387

abstract schema type（抽象模式类型）408

ACID test（ACID测试）472

acknowledgement，参见message acknowledgement

actor playing an entity（扮演实体的演员）350

administered object（受管对象）616

Advice Guy bean（顾问bean）15

afterBegin() 512

afterCompletion() 512

Application（应用）33

  making a new J2EE app in the deploytool（在deploytool中建立一个新J2EE应用）33

Application Assembler（应用组装人员）26

Application Assembler s responsibility（应用组装人员的职责）

  <ejb-link> 615

  deployment descriptor（部署描述文件）617

  security（安全）574, 588

Application exceptions（应用异常）532

application exceptions（应用异常）500, 548

architectural overview（体系结构概述）63

arguments（参数）67

Atomic（原子性）472

attributes，参见 transaction attributes

authentication（认证）572

authorization（授权）572

Auto-acknowledge，自确认 460

# B

Bean-managed transactions（bean托管事务），参见BMT

beanness（bean性）188，191， 196

beans 11，参见entity beans；参见message-driven beans；参见session beans

  bean inheritance（bean继承）186

  five things you do to build a bean（构建bean要做的5件事）16

  overview（概述）98

bean creation（bean创建）188，192，194，196， 参见entity beans

bean getting shot（bean被杀死）283

Bean Provider（bean提供者）26

  exception responsibilities（关于异常的职责）541

  passivation responsibility for stateful session beans（关于有状态会话bean钝化的职责）203

  responsibility for the Deployment Descriptor（关于部署描述文件的职责）617

  security responsibilities（关于安全的职责）589

  writing a CMP entity bean（编写一个CMP实体bean）313

  writing a message-driven bean（编写一个消息驱动bean）447

beat box，参见appendix A

beforeCompletion() 512

begin() 476，478

Being John Entity Bean（作一个John实体bean）350

BETWEEN 416，参见 EJB-QL

BMP（bean托管持久存储）303

BMT（bean容器托管事务）476，478，482，485，487，490，502，508

  getStatus() 489

restrictions（限制）481

summary of bean-managed demarcation（bean托管划分小结）508

bodyguard（保镖）94

bullet points（要点）166

architectural overview（体系结构概述）97

client view（客户视图）131

EJB-QL 420

entity client view（实体客户视图）287

entity relationships（实体关系）420

exceptions（异常）555

local interfaces（本地接口）166

overview（概述）14

Remote interfaces（远程接口）144

RMI（远程方法调用）76

session bean lifecycle（会话bean生命周期）221

business interface（业务接口）79

business methods（业务方法）199, 226

# C

cascade-delete（级联删除）387，388

cast（类型强制转换）120

narrow before cast（类型强制转换之前窄化）120

client（客户）55

client Code（客户代码）57，219

client view（客户视图）111

component interface（组件接口）136

exceptions（异常）159

local（本地）145

code example（代码示例）119

creating a test client（创建一个测试客户）55

InitialContext 117

local client code（本地客户代码）160

lookup（查找）116

Client before and after message-driven beans（有消息驱动bean之前和之后的客户）441

client Jar（客户Jar）51

client view（客户视图）111，131

EJBHome methods（EJBHome方法）129

EJBObject methods（EJBObject方法）137

entity bean（实体bean）287

exceptions（异常）159

local（本地）145

local interface code（本地接口代码）156

CMP 302

BMP 303

CMR 383，396，参见cascade-delete

cmr-field 387

CMR field（CMR字段）383，386

CMT 476，482，485，487，499，502，509

summary of container-managed transactions（容器托管事务小结）509

CMT 491

code examples（代码示例）

AdviceStatefulBean code（AdviceStatefulBean代码）218

client（客户）119

AdviceStatefulBean client（AdviceStatefulBean客户）219

exceptions in client interfaces（客户接口中的异常）538

CMP entity bean complete（完整的CMP实体bean）318

CustomerBeanCMP 318

Customer BMT entity bean（Customer BMT实体bean）268

Customer CMT entity bean，Customer CMT实体bean 316

entity bean Remote home interface（实体bean远程home接口）275

exceptions in client interfaces（客户接口中的异常）538

local client code（本地客户代码）160

local client interfaces（本地客户接口）156

message-driven beans（消息驱动bean）446

MovieBean CMP code（MovieBean CMP代码）396

MovieHomeRemote interface（MovieHomeRemote接口）398

Remote home interface（远程home接口）126

WelcomeBean 446

Coffee Cram（模拟测验）

appendix A: final mock exam（附录A：最后的模拟测验）637

chapter 1: overview（第1章：概述）59

chapter 10: exceptions（第10章：异常）563

chapter 11: security（第11章：安全）593

chapter 12: environment and programming restrictions（第12章：环境和编程限制）627

chapter 3: client view（第3章：客户视图）168

chapter 4: session beans（第4章：会话bean）244

chapter 5: entity bean client view（第5章：实体客户视图）288

chapter 6: entity bean synchronization（第6章：实体bean同步）362

chapter 7: entity relationships and EJB-QL（第7章：实体关系和EJB-QL）425

chapter 8: message-driven beans（第8章：消息驱动bean）463

chapter 9: transactions（第9章：事务）516

commit（提交）471，473

commit() 478

commit options（提交选择）356

compare（比较）

equals() vs. isIdentical() 150

Remote vs. local interfaces（远程接口与本地接口）154

compiling a bean（编译bean）30

component-based development（基于组件的开发）3

component interface（组件接口）87，133，234，271，272

implementing（实现）87

confusingly-named exceptions（名字容易混淆的异常）554

Consistent（一致性）472

constructor（构造函数）195, 196

container（容器）6

container-managed persistence，参见 CMP

container-managed relationship，参见 CMR

Container-managed transactions，参见 CMT

container callbacks（容器回调）181，310

ejbActivate() 183

ejbCreate() 181

ejbPassivate() 183

ejbRemove() 183

implementing in a session bean（在会话bean中实现）183

setSessionContext() 183

Container responsibilities（容器职责）

exceptions（异常）546

create() 123，232，237

entity vs. session bean（实体bean与会话bean）281

CreateException 125，548

create methods（创建方法）159, 181

creation（创建）

entity beans（实体bean）329

entity vs. session bean（实体bean与会话bean）281

session beans（会话bean）188，192

# D

database（数据库）301

database to entity mapping（数据库到实体的映射）400

DataSource 602，610

dead bean（死bean）284

entity vs. instance death（实体死掉与实例死掉）285

declarative security（声明方式实现安全）572

declarative transactions，参见 CMT

DELETE 265

deploy-time customization（部署时定制）602，618

Deployer（部署人员）27

deployment descriptor responsibilities（部署描述文件职责）618

responsibilities for mapping abstract roles to people（将抽象角色映射到人的职责）583

deployment descriptor（部署描述文件）

<ejb-link> 615

abstract persistence schema（抽象持久存储模式）384

abstract schema（抽象模式）394

intro（引言）23

message-driven bean（消息驱动bean）449

resource manager connection factories（资源管理器连接工厂）610

transactions（事务）502，503，504

method parameters（方法参数）505

wildcard（通配符）579

deploytool 32

deserializing（逆串行化）71

Director（导演）378

Director talking to entity bean（导演与实体bean对话）284

distributed transactions（分布式事务）473

two-phase commit（两段提交）473

DuplicateKeyException 548

Dups-ok-acknowledge 460

Durable（持久）472

durable subscription（持久订购）450

# E

EJB

and RMI（和RMI）81

architectural overview（体系结构概述）63

EJBObject 82

ejb-jar 24，620，624

requirements（需求）24

ejb-jar.xml 624

EJB-QL 401，402，411

AND 405

BETWEEN 416

Collection types（Collection类型）413

FROM clause（FROM子句）403

identifiers（标识符）410

IN operator for the FROM clause（FROM子句的IN操作符）415

IN with the WHERE clause（WHERE子句中的IN）417

IS EMPTY 418

IS NOT EMPTY 418

LIKE expression（LIKE表达式）419

parameters（参数）405

path navigation（路径导航）404

reserved words（保留字）410

SELECT clause（SELECT子句）403，406

String literals（String直接量）417

topics and queues（主题和队列）450

using collections in a path（在路径中使用集合）414
WHERE 412
WHERE clause（WHERE子句）403，411
ejb-relation 387
ejb-relationship-role 387
ejbActivate() 183，200，204，207，308，314，354
EJBContext 477
ejbCreate() 181，195，196，237，331，333
EJBException 537，539，552，558
EJBHome 128
methods（方法）129
ejbLoad() 314，354
EJBLocalHome 145
EJBLocalObject 145，156
EJBMetaData 149
EJBObject 82，94，133，137
ejbPassivate() 183，200，204，207，308，314，355
ejbPostCreate() 334，396
ejbRemove() 183，210，213，214，216，314
missed ejbRemove() in session beans（会话bean中没有ejbRemove()调用）213
ejbStore() 314，355
EJB 2.0 guarantees（EJB 2.0保证）
services and behavior（服务和行为）623
supported APIs（支持的API）621
EJB references（EJB引用）613
<ejb-local-ref> 614
EJB Roles and Responsibilities（EJB角色和职责）26
Application Assembler（应用组装人员）26
Bean Provider（bean提供者）26
Deployer（部署人员）27
EJB security mapping（EJB安全映射）590
EntityBean 312，314
entity beans（实体bean）99
activation（激活）352，354
actor playing an entity（扮演一个实体）350
business method（业务方法）348
client view（客户视图）287
CMP bean construction（CMP bean构造）326，327
CMP bean creation（CMP bean创建）328，338
ejbCreate() 331，333
ejbPostCreate() 334
OID for bean creation（bean创建的对象交互图）329
CMP class example（CMP类示例）316
CMP complete class code（完整的CMP类代码）318
container callbacks（容器回调）310，312
ejbActivate() 308
ejbPassivate() 308
ejbPostCreate() 396
entity realization (how a bean becomes an entity)［实体实现（一个bean如何成为一个实体）］306
entity vs. instance death（实体死掉与实例死掉）285
finder methods（查找方法）276，339，341
home business method（home业务方法）345
home interface（home接口）273，275
lifecycle（生命周期）320
mapping from abstract schema to a real database（从抽象模式映射到一个实际的数据库）400
overview（概述）268
passivation（钝化）308，355
relationships（关系）378
unidirectional（单向）391
relationship assignments（关系指派）421

remove() 282，283，284

session remove vs. entity remove（会话删除与实体删除）282

synchronization（同步）298， 301

environment（环境）

creating a subcontext（创建一个子上下文）609

resource mapping（资源映射）612

subcontexts（子上下文）604

environment entries（环境项）607

scope（作用范围）608

types（类型）608

equals() 150

exam objectives（考试大纲）

bean environment（bean环境）600

client view（客户视图）112

EJB overview（EJB概述）2

entity bean client view（实体bean客户视图）260

entity bean lifecycle（实体bean生命周期）296

entity relationships: CMP and CMR（实体关系：CMP和CMR）374

exceptions（异常）526

message-driven beans（消息驱动bean）438

security（安全）570

session beans（会话bean）174

transactions（事务）470

exceptions（异常）555

application exceptions（应用异常）533，548

client interfaces（客户接口）159

common system exceptions（常用的系统异常）551

CreateException 548

DuplicateKeyException 548

EJBException 537，539

FinderException 538，548

logging system exceptions（将系统异常记入日志）535

ObjectNotFoundException 548

RemoteException 78，133，537

RemoveException 548

scenarios（场景）556

system exceptions（系统异常）534

exception pathways（异常之路）530

exception scenarios（异常场景）556

Exercise（练习），参见 Sharpen your pencil

exotic casting（奇特的类型强制转换）122，参见 narrow


# F


findByPrimaryKey() 274

FinderException 538，548，559

finder methods（查找方法）274，276，339，341

Container writes all the code（容器编写所有代码）342

FROM 411


# G


getCallerPrincipal() 241，587

getEJBHome() 137，241

getEJBLocalHome() 241

getEJBLocalObject() 241

getEJBMetaData() 129

getEJBObject() 241

getHandle() 137

getHomeHandle() 129

getPrimaryKey() 137

getRollbackOnly() 241，488，500

getStatus() 489

getUserTransaction() 241，476，477，478

# H

Handle 139, 149

handle（句柄）149

home 91

- business methods（业务方法）278
- create() 123
- EJBHome 128
- home interface（home接口）114
- methods（方法）129

home business method（home业务方法）278， 345

home interface（home接口）273

- EJBHome 128
- entity bean Remote home interface（实体bean远程home接口）273，279
- methods（方法）129
- rules（规则）125

home object（home对象）91

# I

IIOP 90

IllegalStateException 551，552，558，560

IN，参见 EJB-QL

inheritance（继承）186

InitialContext 117

input parameters（输入参数）412，参见 EJB-QL

INSERT 265

instance-level security（实例级安全）586

interface（接口）

- business interface（业务接口）79
- comparing Remote vs. local（比较远程接口和本地接口） 154，163
- component interface（组件接口）82，133，271
- EJBHome 129
- EJBObject 137
- extending（扩展）85
- home interface（home接口）91，114，125，273
- javax.ejb.EntityBean 182
- javax.ejb.EntityContext 182
- javax.ejb.MessageDrivenBean 182
- javax.ejb.MessageDrivenContext 182
- javax.ejb.SessionBean 182
- javax.ejb.SessionContext 182
- local client interfaces（本地客户接口）145，166
- exceptions（异常）159
- Remote（远程）78
- squeamish about not implementing（对不实现感觉不舒服）86

isCallerInRole 587

isCallerInRole() 587

isCallerInRole(String s) 241

isIdentical 141

isIdentical() 142

isIdentical(Object o) 137

Isolated 472

IS EMPTY，参见 EJB-QL

IS NOT EMPTY，参见 EJB-QL

# J

J2EE Reference Implementation (RI) [J2EE参考实现（RI）]

Client Jar（客户Jar）51

deploytool 32

JNDI name（JNDI名）52

New Enterprise Bean Wizard（新建企业bean向导）37

server（服务器）31

verifier（验证器）47

java.net.Socket 625

java:comp/env 602, 603

subcontexts（子上下文）604

JavaMail 621

javax.ejb.EJBContext 477

javax.ejb.EntityBean 312，参见EntityBean

javax.ejb.SessionBean 182，参见SessionBean

javax.ejb.SessionSynchronization 511

javax.jms.QueueConnectionFactory 611

javax.jms.Topic 449

javax.jms.TopicConnectionFactory 611

javax.mail.Session 611

javax.transaction.UserTransaction 471，477

JAXP 621

JDBC 621

JMS 452，621

JNDI 92，602

bean s special context（bean的特殊上下文）189，602

InitialContext 117，602

lookup（查找）116

narrow the result（窄化结果）120

JNDI API 1.2 621

JNDI name（JNDI名）52

JTA 621

JTS 471

# L

LIKE expression（LIKE表达式）

EJB-QL，参见 EJB-QL

local client view（本地客户视图）145

local interface（本地接口）148，156

exceptions（异常）159

vs. Remote interface（远程接口）154

locking (database) [加锁（数据库）] 301

logging（日志）535

lookup（查找）116

# M

Mandatory 492

Many-to-Many（多对多）380

message-driven beans（消息驱动bean）106，437，442

BMT 460

CMT 460，500

concurrent processing（并发处理）444

JMS 452

lifecycle（生命周期）445

MessageDrivenContext 455，456

MessageListener 452

message acknowledgement（消息确认）458，460

queues（队列）454

rules for writing the class（编写类的规则）448

topics（主题）453

transaction attribute（事务属性）500

MessageDrivenContext 455，456

MessageListener 452

message acknowledgement（消息确认）458

methods（方法）

afterBegin() 512

afterCompletion() 512

beforeCompletion() 512

begin() 476

create() 128，274

create methods（创建方法）159

ejbActivate() 183，200，204，207，309，354

ejbCreate() 183，195，331，333

ejbLoad() 354

ejbPassivate() 183，200，204，207，309

ejbPostCreate() 334

ejbRemove() 213，214，216

findByPrimaryKey() 274

getCallerPrincipal() 587

getEJBHome() 137

getEJBMetaData() 129

getHandle() 137

getHomeHandle() 129

getPrimaryKey() 137

getRollbackOnly() 488

getStatus() 489

getUserTransaction() 476

isIdentical(Object o) 137，141

remove() 137，282

remove(Handle h) 129

remove(Object key) 129

remove methods（删除方法）159

setRollbackOnly() 485

setSessionContext() 183

setSessionContext(SessionContext sc) 195

method permissions（方法许可）578

<unchecked/> 581

union（并集）580

missed ejbRemove() in session beans（会话bean中没有ejbRemove()调用）213，215

mock exam，参见Coffee cram

Movie（电影）378

MovieBean code（MovieBean代码）396

MovieHomeRemote interface（MovieHomeRemote接口）398

MovieSchema 408

multiplicity（多重性）380，387，422

affects return type（影响返回类型）382

Many-to-Many（多对多）380

One-to-Many（一对多）380

One-to-One（一对一）380


# N


narrow（窄化）120

PortableRemoteObject.narrow() 121

network transparency（网络透明性）65，66

Never 492，495

NoSuchEntityException 551，552

NoSuchObjectException 551，553

NoSuchObjectLocalException 551，552，553

NotSupported 492，495

NOT IN，参见 EJB-QL

# O

objectives（大纲），参见 exam objectives
ObjectNotFoundException 548，553，559
home business method（home业务方法）346
One-to-Many（一对多）380
One-to-One（一对一）380

# P

passing objects remotely（远程传递对象）68
passivation（钝化）
  stateful session bean（有状态会话bean）200
  transactions（事务）206
pass by copy（传副本）69
pass by reference（传引用）68
path navigation（路径导航）404
persistent field（持久字段）313
  virtual（虚）315
point-to-point（点到点）451，参见queue；参见topic
PortableRemoteObject.narrow() 121
primary key（主键）332，333
Principal（主体）584
programmatic security（程序方式实现安全）586，589
programmatic transactions，参见 BMT
programming restrictions（编程限制）625
propagation（传播）
  transactions（事务）474，482
proxy（代理）参见 stub
publish/subscribe（发布/订购）451

# Q

queries（查询）406，参见 EJB-QL
queues（队列）450，454

# R

relationships（关系）378，394
  unidirectional（单向）391
relationship assignments（关系指派）421
Remote（远程）65
  arguments（参数）67，72
  Serializable 72
  business interface（业务接口）79
  component interface（组件接口）82
  home interface（home接口）125
  interface（接口）78
  PortableRemoteObject.narrow() 121
  RemoteException 78
  return values（返回值）67, 72
  Serializable 72
  vs. local interface（本地接口）154
RemoteException 78，133，537，559
Remote home interface（远程home接口）125，279
  EJBHome 128
Remote interface（远程接口）78，144
  component interface（组件接口）82，137，271
  home interface（home接口）125
  EJBHome 128
  vs. local interface（本地接口）154
Remote Method Invocation（远程方法调用）63，64

Remote object（Remote对象）65，67

EJBObject 82

remove（删除）129，151，159，282

entity beans（实体bean）284

entity vs. session bean（实体bean与会话bean）282，283

remove() 137，151，208，282，286

entity vs. session bean（实体bean与会话bean）283

remove(Handle h) 129

remove(Object key) 129

RemoveException 548，559

remove methods（删除方法）159

Required 492

RequiresNew 492

resource manager connection factory（资源管理器连接工厂）610

resource authorization（资源授权）611

return values（返回值）67

Serializable 72

RMI 63

arguments（参数）67

how its used by EJB（EJB如何使用）81

IIOP 90

PortableRemoteObject.narrow() 121

RemoteException 78

Remote object（Remote对象）65

return values（返回值）67

RMI-IIOP 90

rmic 90

skeleton（骨架）65，67

stub（桩）64

rmic 90

Role（角色）584

role reference（角色引用）589

rollback（回滚）473

rollback() 478

RPC 67

run-as 591

RuntimeException 545


# S


scalable（可扩展）

stateless session beans（无状态会话bean）103

security（安全）

App Assembler（应用组装人员）574，576

authentication（认证）572

authorization（授权）572

context propagation（上下文传播）592

declarative security（声明方式实现安全）572

Deployer（部署人员）583

identity（身份）591

instance-level security（实例级安全）586

isCallerInRole() 587

Principal（主体）584

programmatic security（程序方式实现安全）586

roles（角色）574

role definition（角色定义）576

security identity（安全身份）592

security mapping（安全映射）590

security roles（安全角色）574，576

<method-permission> 578

SELECT 265，402，406

Serializable 67，72

server（服务器）31

Server dude at the pool（在bean池“享乐”的服务器）441

SessionBean 182，183

SessionContext 189，199，240，241

SessionSynchronization 511，512

callbacks（回调）512

session beans（会话bean）182

bean lifecycle（bean生命周期）180，184

container callbacks（容器回调）182

implementing（实现）183

missed ejbRemove() calls（没有得到ejbRemove()调用）215

overview（概述）98，100

passivation（钝化）200，207

removal（删除）208，210

scalability（可扩展性）103

SessionBean interface（SessionBean接口）182

stateful session beans（有状态会话bean）178，184

creation（创建）179，192，196

OID for bean creation（bean创建的对象交互图）194

passivation/activation（钝化/激活）207

removal（删除）210

stateless session bean（无状态会话bean）102，222

creation（创建）

OID for bean creation（bean创建的对象交互图）227

session context（会话上下文）240

setEntityContext（EntityContext ec） 314

setRollbackOnly() 485，487

setRollbackOnly(boolean b) 241

setSessionContext(SessionContext ctx) 183

setSessionContext(SessionContext sc) 195

Sharpen your pencil

APIs 621

architectural overview（体系结构概述）107

bean table（bean表）108

bean types（bean类型）13

client view exercise（客户视图练习）165

CMP entity creation（CMP实体创建）359

compiler checked? （编译器是否检查？） 447

EJB-QL 407

EJB architecture（EJB体系结构）10， 80

EJB Lifecycle（EJB生命周期）96

EntityBean interface（EntityBean接口）323

entity bean（实体bean）280

entity bean callbacks（实体bean回调）353

entity bean class methods（实体bean类方法）361

entity bean relationships (Director-to-Movie)［实体bean关系（Director-Movie）］390

entity business method（实体业务方法）360

entity CMP code annotation（实体CMP代码注解）319

entity lifecycle activity（实体生命周期活动）324

environment（环境）626

exceptions（异常）561

exception exercise（异常练习）562

exception scenarios（异常场景）557

home interface（home接口）132

local and Remote client interfaces（本地和远程客户接口）155

local clients（本地客户）158

local client view（本地客户视图）147

message-driven bean（消息驱动bean）461

message-driven bean context methods（消息驱动bean上下文方法）455

Multiplicity and Assignments（多重性和指派）423

package name（包名）119

security roles（安全角色）575

SessionBean interface（SessionBean接口）186

session bean overview（会话bean概述）104

stateful session bean state diagram（有状态会话bean状态图）187

stateless to stateful（无状态到有状态）220

transactions（事务）480

transactions puzzle（事务填空题）514

transaction attributes（事务属性）496，497，507

UserTransaction 484

write an entity bean（编写实体bean）317

skeleton（骨架）65，67

squeamish（不舒服）86

stateful session bean（有状态会话bean）178

creation（创建）179，194，196

Object Interaction Diagram (OID) for bean creation（bean创建的对象交互图）194

removal（删除）210

SessionSynchronization 512

stateless session（无状态会话）

transactions in（其中的事务）481

stateless session bean（无状态会话bean）102，222

business methods（业务方法）226

creation（创建）224

Object Interaction Diagram (OID) for bean creation（bean创建的对象交互图）227

String literals（String直接量）参见 EJB-QL

stub（桩）64，67

subcontexts（子上下文）604

supported APIs（支持的API）621

Supports（支持）492，495

suspended transactions（挂起的事务）483

synchronization（同步）298

system exceptions（系统异常）500，551

logging（记入日志）535


# T


The Advice Guy（顾问）15

Tikibean Lounge（bean的对话）

BMT/CMT 501

EJBObject/bean 89

session bean/bartender（会话bean/酒吧招待）14

topics（主题）450，453

durable subscription（持久订购）450

topics and queues（主题和队列）450

TransactionRequiredException 551，552

TransactionRequiredLocalException 551，552

TransactionRolledbackException 551，552

TransactionRolledbackLocalException 551，552

transactions（事务）471,473，参见commit，参见rollback，参见setRollbackOnly()

Bean-managed transactions（bean托管事务）476

begin() 476

BMT 476，481

call stack example（调用栈示例）479

database locking（数据库加锁）301

deployment descriptor（部署描述文件）503

getRollbackOnly() 488，500

message-driven beans（消息驱动bean）460，500

propagation（传播）474， 482

SessionSynchronization 512

stateful session bean passivation（有状态会话bean钝化）206

suspension（挂起）475，483

transaction attributes（事务属性）492

two-phase commit（两段提交）473

unspecified transaction context（未指定的事务上下文）499

transaction attributes（事务属性）492，493，498

attributes that require a transaction（需要事务的属性）494

Mandatory 492

Never 492，495

NotSupported 492，495

Required 492

RequiresNew 492

Supports 492，495

Tutorial（教程）28

two-phase commit（两段提交）473

# U

unidirectional（单向）391

unsetEntityContext() 314

unspecified transaction context（未指定的事务上下文）499

UPDATE 265

Users（用户）584

UserTransaction 476，477

getStatus() 489

# V

vendor lock-in（被开发商限制住）5

verifier（校验器）47

virtual persistent fields（虚持久字段）315

# W

WHERE 403，411，412

wildcard（通配符）

method elements（方法元素）504

transaction attributes（事务属性）504